## Міністерство освіти і науки Украіни

### Національний технічний університет України "Київський політехнічний інститут імені Ігоря Сікорського"

Фізико-технічний інститут

## Криптографія

Лабораторна робота №4

Виконали студенти групи ФБ-13 Дмитрів Анастасія Лагно Костянтин

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

**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.
- 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 i 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 М 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.

#### Хід роботи:

1. Написали функцію пошуку випадкового простого числа з заданого інтервалу або заданої довжини, використовуючи датчик випадкових чисел та тест перевірки на простоту. В якості теста перевірки на простоту, вибрали тест Міллера-Рабіна. Оскільки даний тест використовує пошук НСД, з попередньої лабораторної взяли функції gcd та inverse\_mod.

#### Код функцій:

```
def gcd(a, b):
    while b:
    a, b = b, a % b
    return a
```

```
#a^-1
def inverse_mod(a, mod, v = None):
    if v is None:
       v = [0, 1]
    if a == 0 or gcd(a, mod) != 1:
       return None
    else:
        d = mod % a
        q = mod // a
        v.append((v[len(v) - 2] - q * v[len(v) - 1]))
        if d != 0:
            mod, a = a, d
            return inverse_mod(a, mod, v)
        else:
            return v[len(v) - 2]
def miller_rabin(n):
    k = 100
    if n <= 1:
       return False
    if n <= 3:
       return True
    # Розклад n - 1 = 2^s * d, де d непарне
    s, d = 0, n - 1
    while d % 2 == 0:
        s += 1
        d / = 2
    for _ in range(k):
        x = random.randint(1, n)
        if gcd(n, x) != 1:
           return False
        xd = pow(x, d, n)
        if xd == 1 or xd == n - 1:
            return True
        for i in range(0, s-1):
            w = (2**i)*d
            if pow(x, w, n) == n-1:
               return True
    return False
```

Перевірка роботи буде в наступному пункті, оскільки вони пов'язані.

2. З допомогою даних функцій, а також функцій generate\_random\_prime і generate\_prime\_pair згенерували дві пари чисел довжиною 256 біт. Зразу також додали перевірку, якщо пара для абонента А більше, ніж для В, то вони міняються місцями, замість того щоб заново генеруватись.

Код функцій:

```
def generate_random_prime(bits):
    while True:
        number = random.randint(2**(bits-1), 2**bits - 1)
        if miller_rabin(number):
            return number

def generate_prime_pair(bits):
    p = generate_random_prime(bits)
    q = generate_random_prime(bits)
    p1 = generate_random_prime(bits)
    q1 = generate_random_prime(bits)
```

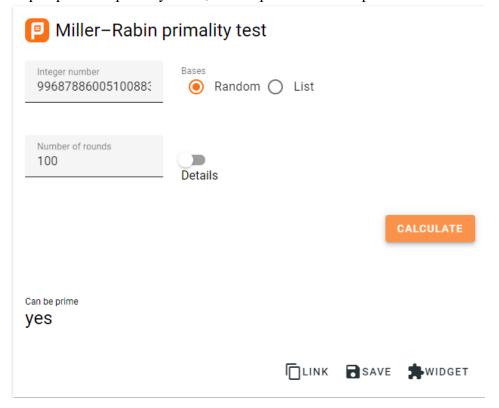
```
if p*q > p1*q1:
    p, p1 = p1, p
    q, q1 = q1, q
return p, q, p1, q1
```

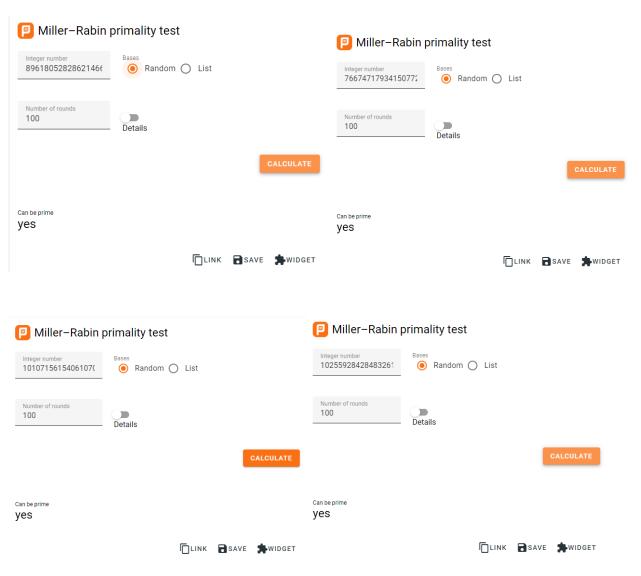
#### Перевірка роботи:

Якщо число  $\epsilon$  простим, функція тесту виводить значення True, в іншому випадку — False. Функції для генерації числа та пари чисел мають виводити прості числа розміром 256 байт, якщо ж числа не прості — функція циклічно генеру $\epsilon$  нове число, поки всі не будуть простими.

Бачимо, що коли в функцію тесту передається число 7, яке  $\epsilon$  простим, результатом  $\epsilon$  True, а при 10, яке не  $\epsilon$  простим, False.

Перевірю на окремому сайті, чи є отримані числа простими:





I окремим скриптом перевірю числа на розмір:

Як бачимо, функції працюють коректно.

```
3 - 4 - 5.
```

Тепер переходимо до основного завдання лабораторної. Код основних функцій:

```
def GenerateKeyPair(p, q):
    n = p*q
    f = (p-1)*(q-1) #ойлера
    e = (2**16) + 1
    if gcd(e, f) != 1:
        return false
    else:
        m = inverse_mod(e, f)
```

```
if m < 0:
             d = m + f
             d = m
         public_k = (e, n)
         secret_k = (d, n)
         return public_k, secret_k
def Encrypt(key_p, M): #tuple
    e, n = \text{key\_p[0]}, \text{key\_p[1]}
    C = pow(M, e, n)
    return C
def Decrypt(key_s, C):
    d, n = \text{key}_s[0], \text{key}_s[1]
    M = pow(C, d, n)
    return M
def Sign(key_s, k):
    d, n = \text{key}_s[0], \text{key}_s[1]
    S = pow(k, d, n)
    return S
def Verify(k, S, key_p):
    e, n = \text{key\_p[0]}, \text{key\_p[1]}
    return k == pow(S, e, n)
def SendKey(key_s, key_p):
    e1, n1 = \text{key}_p[0], \text{key}_p[1]
    k = random.randint(1, 1000)
    k1 = pow(k, e1, n1)
    S = Sign(key_s, k)
    S1 = pow(S, e1, n1)
    return k1, S1, k
def ReceiveKey(key_s, k1, S1, key_p):
    d1, n1 = key_s[0], key_s[1]
    k = pow(k1, d1, n1)
    S = pow(S1, d1, n1)
    check = Verify(k, S, key_p)
    return check
```

#### Генеруємо дві пари ключів для кожного з абонентів:

#### Шифрування повідомлень:

```
Encryption
Message for B:
776
Encrypted text by A for B:
936581553507222591531980607525346914036704409349240904891857587052026872569453955952779184687247567117895878050080428343897805486616384830134509733048151

Message for A:
400
Encrypted text by B for A:
3266625840892765578109633059707590961128250100494717811730465124161578580075867574949480189967431021399852126028636460489263311588414196336039842534959299
```

Розшифрування повідомлень:

```
Decrypted text by A from B:
400
Origin B`s message:
400

Decrypted text by B from A:
776
Origin A`s message:
776
```

Обмін ключами (тут одразу і використовується створення цифрового підпису):

```
Keys trading
A generate k1 and S1
k1: 8002012342447716377838432616740040251747738553528504307050055820462038963731831168660036380246503019400708396987225765337112772334423448573188044936928107
S1: 912032943509747625381949307163979650124102137334572277749322704806377530704787614996872134391143959993647142510158588278773940000458528076856486981885646
A sends message (k1, S1) to B ------> B recieved message
B verifies the signature
Has it been verified? True

B generate k2 and S2
k2: 1902082132203651102207208370526465834311350451537912620087314442351710848355488400570230527184686479122772942912444016813954011472320531974423596513339672
S2: 5788673539922657056451184178556240933757105090149702936331763447444491653797010063353846301515541162840014678040693082589637645511691959900088580150200084
B sends message (k2, S2) to A ------> A recieved message
A verifies the signature
Has it been verified? True
```

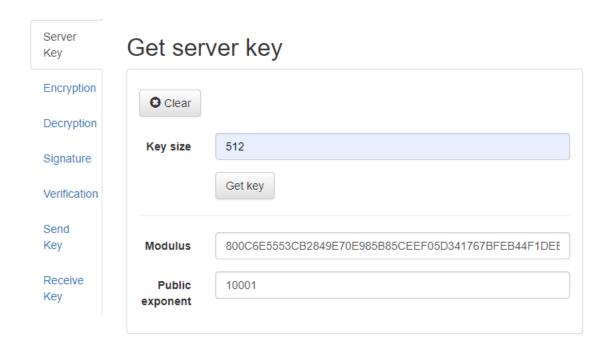
Тепер перевіримо роботу нашої програми з сервісом з методички. Основний момент, над яким ми довго ламали голову — в нас працювали всі функції і завдання, окрім отримання ключа сайтом. Як виявилось, на сайті потрібно при генерації ключів задавати вдвічі більше значення розміру, ніж яке ми задаємо при створенні ключів у нас, так як якщо задати на сайті 256 біт, і у нас 256, то виходить, що у сайту значення п буде 256 біт, а у нас, так як ми перемножуємо числа розміром 256 байтів, набагато більше. На роботу шифрування, дешифрування і перевірки цифрового підпису це ніяк не впливає, оскільки там допускається різниця між розмірами ключів в абонентів в обидві сторони, а от при розсиланні ключів важливо, щоб розмір п1 у абонента, який отримує ключ, був більшим, ніж розмір п у того, хто надсилає. Додатково ще була додана перевірка, що якщо публічний ключ нашого абонента більше, ніж на сайті, то пари ключів генеруються заново, поки публічний ключ сайту не буде більшим:

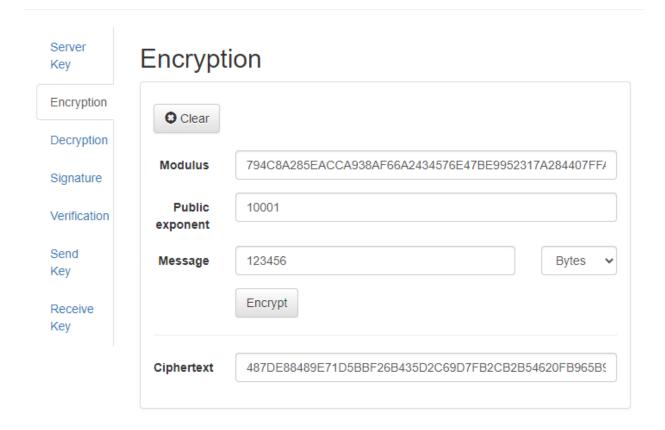
```
#надсилаємо свій ключ та сигнатуру серверу
print('Sending Key and S for server')

while Public_A[1] > public_server[1]:
    a = generate_prime_pair(256)
    Public_A, Secret_A = GenerateKeyPair(a[0], a[1])[0], GenerateKeyPair(a[0], a[1])[1]

print('New keys of A: ')
print('Public_A (e, n): ', '\n e: ', decimal_to_hex(Public_A[0]), '\n n: ', decimal_to_hex(Public_A[1])_)
print('')
```

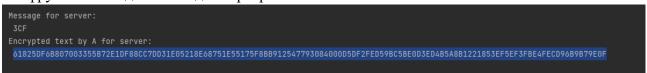
Створення пари і розшифрування повідомлення, отриманого від сайту:

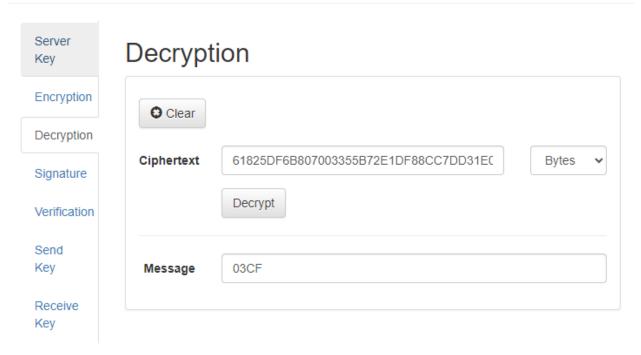




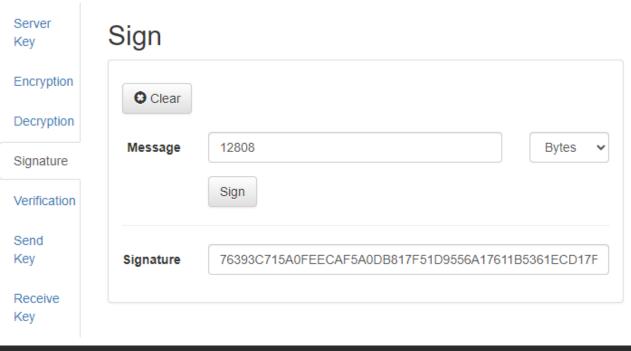
| Checking using the server  |
|--|
|  |
| Public_A:  |
| e: 10001   |
| n: 794C8A285EACCA938AF66A2434576E47BE9952317A284407FFA0E797CBD98AFB26EBDFD3C8E5D5529FFADABE51DA6DE78D09F29E90958646FFB9B9B0C3D1CE01                            |
|  |
| Secret_A:  |
| d: 64947616863744358441937859739103972878706763495689744454561437831408822496381434510082595693096940128648402975473604278928145812596950844761184781709273    |
| Please enter server exponent: 10001  |
| Please enter server modulus: 8880c655553082849E70E985885CEEF0503417678FE844F10EE1428F9558678E9A3E58898E3210285AEA6E93A6874CEECEF4EE0C3CC85A9CC46EF2F8681C160F  |
| Encrypted message from server: 4870E88489E710588F26843502C6907F82CB2854620F8965899306F0E64F777AF281436F42209A4972E697431E8F08254121598A08DEE4E5088F7F7A2FBA585 |
| Decrypted message by A: 123456   |
|  |

Шифрування повідомлення для сервера:

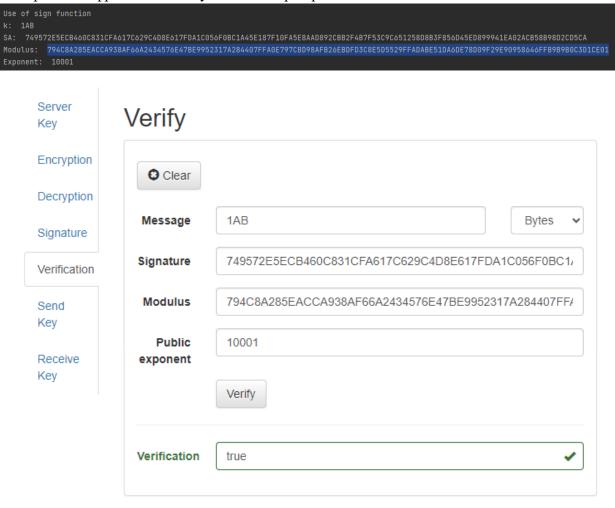




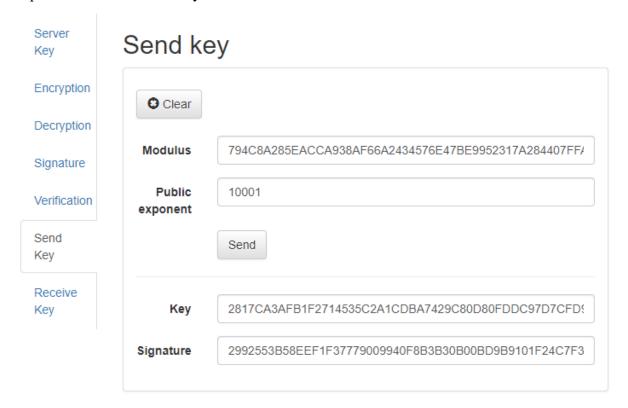
#### Створення цифрового підпису сайтом:



Use of verify function Message from server: 12 Signature from server: Verify? True Створення цифрового підпису нами та перевірка на сайті:



#### Отримання ключа від сайту:



```
Server send key and we try to verify it
k from server: 2817CA3AFB1F2714535C2A1CD8A7429C8080FDDC97D7CFD9EE9A71CF0E6846DFC883D8C8988643DE0A530A9EEE9EC2BC480A788876875EED119711281947043
S from server: 2992553858EEF1F37779089940F8838308008D989101F24C7F30895E888DA67F3AV66E40C6A82FF78B33762B774DF82670384803C829E2550C35655096126178
A verifies the server signature
Has it been verified? True
```

#### Надсилання ключа на сайт:

```
Sending key and S for server

New keys of A:

Public_A (e, n):

e: 10001

n: 794C8A285EACCA938AF66A2434576E47BE9952317A284407FFA0E797CBD98AFB26EBDFD3C8E5D5529FFADABE51DA6DE78D09F29E90958646FFB9B9B0C3D1CE01

A generate k1 and S1

k1: 30D230D1C70CC736389D72613D937EA3E51A0E0F51CAC54A93972620B0FCD569DAA6AB6D1126A986E00385604EDE8451E0C9A33E92766F10D61A04EE527B44DF

S1: 76A2C8D0D182A46748AB1016CA0603AEFACB42975AB8699A4A85597895CD4E3F01DD1F034BB2297AD839B3E404EF01459D2D68D9E093C1BC650BC276B4526919

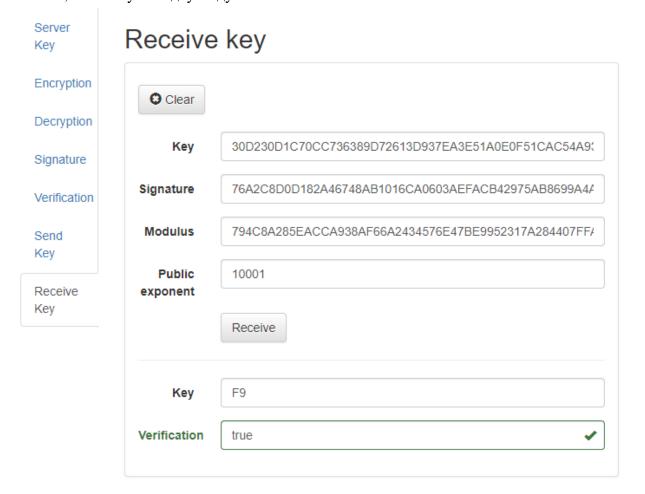
Modulus: 794C8A285EACCA938AF66A2434576E47BE9952317A284407FFA0E797CBD98AFB26EBDFD3C8E5D5529FFADABE51DA6DE78D09F29E90958646FFB9B980C3D1CE01

Exponent: 10001

Key (for checking): F9

Process finished with exit code 0
```

Бачимо, що модуль (відкритий ключ) не змінився, отже виконується умова розсилання ключів, в іншому випадку модуль би змінився на новий.



Верифікація пройдена, отже все працює правильно.

#### Висновки:

В ході виконання даної лабораторної роботи, ми ознайомились з криптосистемою RSA, алгоритмом цифрового підпису та з методами генерації параметрів для асиметричних криптосистем. Під час виконання роботи набули практичних навичок роботи з даними

алгоритмами, розібрали принципи та умови даної криптосистеми, а також отримали додаткові знання по алгоритму пошуку простих чисел.