Протокол лабораторної роботи №4

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

Варіант №7

Виконав

Студент 3 курсу

Групи ФБ-13

Короткевич Іван

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.

Код програми:

Створення 256 битного числа та перевірка числа на простосту:

```
#Generate random 526 bit number
def number_256bit():
    random_number = secrets.randbits(256)
    return random_number


#Function to check if number is prime
def check_prime(p, k=15):

    if p % 2 == 0 or p % 3 == 0 or p % 5 == 0 or p % 7 == 0:
        return False

    s = 0
    d = (p - 1)
    while d % 2 == 0:
        d //= 2
        s += 1

    for i in range(k):
        x = randrange(2, p - 1)

        if gcd(x, p) > 1:
            return False

        a = pow(x, d, p)
        if a == 1 or a == -1:
            continue

        for _ in range(1, s):
            a = pow(a, 2, p)

            if a == p - 1:
                break

        else:
            return False

    return True
```

Знаходження простого чилса:

```python
def select_prime(number):

    if number % 2 == 0:
        x = number + 1
    else:
        x = number

    i = 1
    while i < number / 2:
        if check_prime(x) == True:
            return x

        x += 2*i
        i += 1

    return x
```

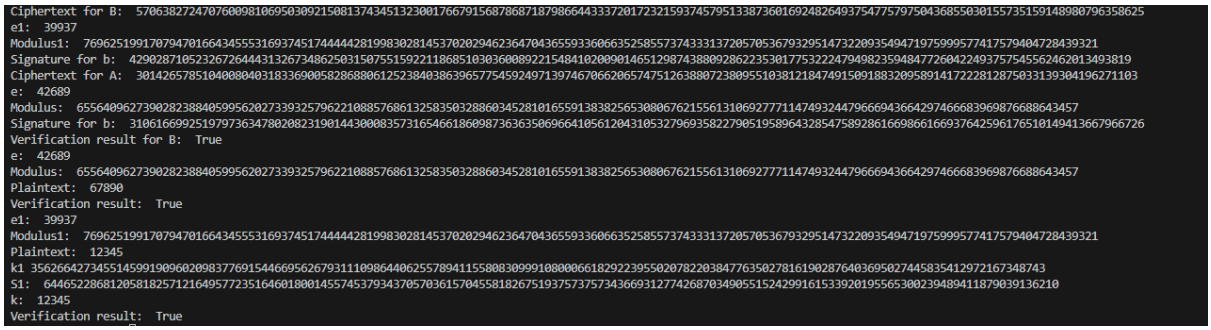
Шифрування, дешифрування, створення підпису та перевірка повідомлення:

```python
def GenerateKeyPair(p, q):
    n = p*q
    phi_n = (p-1)*(q-1)
    exp = secrets.randbits(16)
    e = select_prime(exp)
    d = inverse_elem(e, phi_n)
    return n, e, d


#Encrypt message
def Encrypt(M, e, n):
    C = pow(M, e, n)
    return C


#Sign the message
def Sign(M, d, n):
    S = pow(M, d, n)
    return S


#Descrypt the message
def Decrypt(C, d, n):
    M = pow(C, d, n)
    return M


#Verify integrigty of the message
def Verify(M, S, e, n):
    if M == pow(S, e, n):
        return True
    return False
```

Відправлення та отримання ключа:

```python
#Function that sends common secret
def SendKey(k, d, e1, n, n1):
    k1 = pow(k, e1, n1)
    S = pow(k, d, n)
    S1 = pow(S, e1, n1)
    return k1, S1


#Function that recieves common secret
def RecieveKey(k1, S1, d1, n1):
    k = pow(k1, d1, n1)
    S = pow(S1, d1, n1)
    return k, S
```

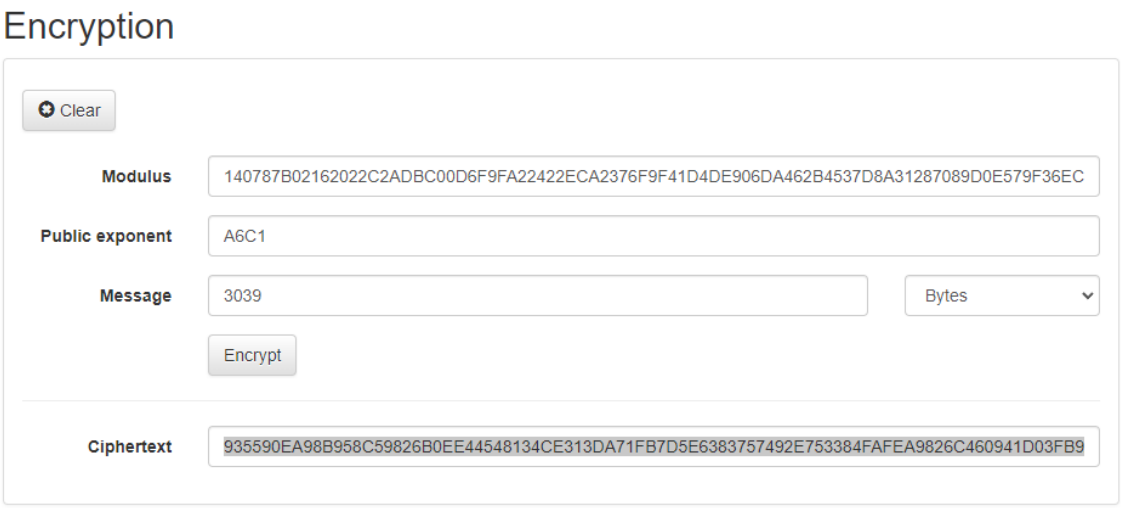
Приклад роботи програми:



Програма може зберігати ключі у спеціальному файлі keys.json

Перевірка роботи:

Для перевірки роботи значення треба було перевести в 16-ти систему счислення:



```
Clinet public(n) and private key:  655640962739028238840599562027339325796221088576861325835032886034528101655913838256530806762155613106927771147493244796669436642974666839698766886434571706180855739854460231024510895479531090027893134145205720766551326471147436809394521419783989608808817955092708495859435872291192155684641200612151942 5
Message from server:  0x3039
```

Decryption

Clear

Ciphertext 8F859C04FBBD2C786A261D1D4A08214EEA4B2F8D90E369BA5669641AF08442 Bytes

Decrypt

Message 3039

```
Message for server:  0x3039
Encrypted message for server:  649168048138037114100435319712336611447658475218994490301677184279598326256865
```

Enter decimal number

649168048138037114100435319712336611 10

= Convert × Reset ⇅ Swap

Hex number (64 digits)

8F859C04FBBD2C786A261D1D4A08214EE
A4B2F8D90E369BA5669641AF0844216 16

Sign

Clear

Message 3039 Bytes

Sign

Signature B3F54154177B0EA2DCE2B69C7499C2A727F7C39AC50F5D540055B3122795EE41

Enter hex number

B3F54154177B0EA2DCE2B69C7499C2A727 16

= Convert × Reset ⇅ Swap

Decimal number (77 digits)

813973283078962545998265008375472
2299660498569716149883652999213175917375041 10

Decimal from signed 2's complement

```
Message signature from server:  813973283078962545998265008375472229966049856971614988365299992131 75917375041
Verification result:  True
```

```
Signature from client:  168368620832669654161752609391881697237433241242949257251680751485234300504713673892815146929545370345031679704262836504162374015403692913962051617141 97
Message:  0xbdf
Client public key: 42689 655640962739028238840599562027339325796221088576861325835032886034528101655913838256530806762155613106927771147493244796669436642974666839698766886434 57
```

Enter decimal number

42689 10

= Convert × Reset ⇅ Swap

Hex number (4 digits)

A6C1 16

Enter decimal number

65564096273902823884059956202733932 10

= Convert × Reset ⇅ Swap

Hex number (127 digits)

140787B02162022C2ADBC00D6F9FA224
22ECA2376F9F41D4DE906DA462B4537D 16

Enter decimal number

168368620832669654161752609391881697 10

= Convert × Reset ⇅ Swap

Hex number (126 digits)

524BFF8330311BF871313DEF81F8FE029
389DE088422FE9CFDB9623C3023E0F2E7 16

Verify

Clear

Message BDF Bytes

Signature 524BFF8330311BF871313DEF81F8FE029389DE088422FE9CFDB9623C3023E0F2E72EDF55A7E702B913B393

Modulus 140787B02162022C2ADBC00D6F9FA22422ECA2376F9F41D4DE906DA462B4537D8A31287089D0E579F36EC

Public exponent A6C1

Verify

Verification true ✔

Send key

Clear

Modulus 140787B02162022C2ADBC00D6F9FA22422ECA2376F9F41D4DE906DA462B4537D8A31287089D0E579F36EC

Public exponent A6C1

Send

Key 0139DF22D672353D6F3A07568E9A869C479D0E8E688CB9184ABD4708ECE052A4293FAB526D9EBEF0CB89

Signature EFB9EE0FED86BA2DF0AC4411295022B633D549C3DA3CFBA05D413D131119CE73776390E6F391E1DCB599

```
Server key and signature:  592764098470254173 8 120671610129643122975372220114192503138937913940020940732941030302841236623 32
Client private key:  17061808557398544602310245108954795310900278931341452057207665513264711474368093945214197839896088088179559027084958594350722911921556846412006121519425
Server public key 65537 96616247683602825070798563963110556653091141416585574575630299175132795599309
Verification result:  True
```