МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»
ФІЗИКО-ТЕХНІЧНИЙ ІНСТИТУТ

ЛАБОРАТОРНА РОБОТА №4

З дисципліни «Криптографія»
Варіант 1

Виконали:
студенти 3 курсу ФТІ
групи ФБ-93
Абдуллаєва Есміра
Шовак Мирослав

Викладач:
Селюх П. В.

м.Київ – 2021

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.

Завдання

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 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) , (e , n) та секретні d і d
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 $0 < k < n$.

Хід роботи

В ході роботи , перш за все , ми почали розбиратися як створити перевірку великого числа на простоту. Реалізували ми це за допомогою метода Міллера-Рабіна , який був зображений у методичних вказівках. Наступним кроком ми приступили до вивчення методу RSA і реалізували його за допомогою формул з того самого джерела. У нашій роботі зображене шифрування не тільки числа , як нам вказувалося у завданні , ми ще змогли реалізувати шифрування тексту завдяки конвертування тексту в hex , а потім в int. Крім того були зроблені 2 функції: Abonent та verify_site для зрозумілішого аналізу отриманих результатів в роботі та перевірки результатів з сайтом.

Код програми

```
from random import randrange
from math import gcd

alphabet = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p',
'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z']


def print ():
    print(
        '+-------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------+')


def euclid ext(a, n):
    if n == 0:
        return a, 1, 0
    else:
        gcd1, x, y = euclid_ext(n, a % n)
        return gcd1, y, x - y * (a // n)


def miller_rabin(num):
    if num % 2 == 0 or num % 3 == 0 or num % 5 == 0 or num % 7 == 0:
        return False
    d = num - 1
    s = 0
    while d % 2 == 0:
        d = d // 2
        s += 1
    x = randrange(2, num)
    if gcd(x, num) > 1:
        return False
    x = pow(x, d, num)
    if x == 1 or x == num-1:
        return True
    for r in range(1, s-1):
        x = pow(x, 2, num)
        if x == num-1:
            return True
        if x == 1:
            return False
    return False


def gen number(bit):
    num = (randrange(1 << bit - 1, 1 << bit) << 1) + 1
    if miller_rabin(num):
        return num
    else:
        num = gen number(bit)
        return num


def gen pairs():
    p, q, p1, q1 = gen number(256), gen number(256), gen number(256), gen number(256)
    while p*q > p1*q1:
        p, q, p1, q1 = gen number(256), gen number(256), gen number(256), gen number(256)
    return p, q, p1, q1


def GenerateKeyPairs(p, q):
    n = p * q
    fi = (p - 1) * (q - 1)
    e = randrange(2, fi)
    while gcd(e, fi) != 1:
        e = randrange(2, fi)
    gcd1, x, y = euclid ext(e, fi)
    d = (x % n + n) % n
    return [n, e], [d, p, q]


def Encrypt(m, e, n):
    return pow(m, e, n)
```

```python
def Decrypt(c, d, n):
    return pow(c, d, n)


def Sign(m, d, n):
    return pow(m, d, n)


def Verify(s, m, e, n):
    return pow(s, e, n) == m


def SendKey(k, e1, d, n1, n):
    print(f'{k = }')
    k1 = Encrypt(k, e1, n1)
    S = Sign(k, d, n)
    print(f'{S = }')
    S1 = Encrypt(S, e1, n1)
    print(f'{k1 = }\n{S1 = }')
    print ()
    return k1, S1


def ReceiveKey(k1, S1, e, d1, n1, n):
    k = Decrypt(k1, d1, n1)
    S = Decrypt(S1, d1, n1)
    print(f'{k = }\n{S = }')
    if Verify(S, k, e, n):
        print('message =', k)
        print('True')
    else:
        print('False')


def int message():
    m = input('Enter your message: ')
    msg = (m.encode('utf-8'))
    msg = int(msg.hex().upper(), 16)
    return msg


def Abonent(p, q, p1, q1):
    print_()
    # A
    public_key, secret_key = GenerateKeyPairs(p, q)
    print(f'Subscriber A:\nn = {public_key[0]}\ne = {public_key[1]}\nd = {secret_key[0]}')
    # B
    public_key1, secret_key1 = GenerateKeyPairs(p1, q1)
    print(f'Subscriber B:\nn = {public key1[0]}\ne = {public key1[1]}\nd = {secret key1[0]}')
    print_()

    message = 1234567890
    k1, S1 = SendKey(message, public key1[1], secret key[0], public key1[0], public key[0])
    ReceiveKey(k1, S1, public_key[1], secret_key1[0], public_key1[0], public_key[0])
    print ()
    # A
    message = int_message()
    print(f'Message in int view: {message}')
    mass_m = Encrypt(message, public_key[1], public_key[0])
    mass d = Decrypt(mass m, secret key[0], public key[0])
    msg = bytearray.fromhex(hex(mass d)[2:]).decode()
    mass_s = Sign(message, secret_key[0], public_key[0])
    mass v = Verify(mass s, message, public key[1], public key[0])
    print('\nDecrypted message: ', mass_m, '\nEncrypted message: ', msg, '\nSignature: ',
mass s, '\nVerify: ', mass v)
    print_()

    # B
    message = int_message()
    print(f'Message in int view: {message}')
    mass_m = Encrypt(message, public_key1[1], public_key1[0])
    mass d = Decrypt(mass m, secret key1[0], public key1[0])
    msg = bytearray.fromhex(hex(mass_d)[2:]).decode()
    mass_s = Sign(message, secret_key1[0], public_key1[0])
    mass_v = Verify(mass_s, message, public_key1[1], public_key1[0])
```

```
    print('\nDecrypted message: ', mass m, '\nEncrypted message: ', msg, '\nSignature: ',
mass_s, '\nVerify: ', mass_v)
    print ()


def verify site decrypt(hex e, hex n):
    demical_modulus = int(hex_n, 16)
    demical public exp = int(hex e, 16)

    msg = int_message()

    hex_msg = hex(Encrypt(msg, demical_public_exp, demical_modulus))[2:].upper()
    return msg, demical modulus, demical public exp, hex msg


def verify_site_encrypt(e, n):
    msg = int message()

    hex n = hex(n)[2:].upper()
    hex e = hex(e)[2:].upper()

    int msg = Encrypt(msg, e, n)
    hex_msg = hex(int_msg)[2:].upper()

    return n, e, msg, hex n, hex e, hex msg


def verify_site_sign(hex_e, hex_n, sign):
    demical modulus = int(hex n, 16)
    demical public exp = int(hex e, 16)
    demical_sign = int(sign, 16)

    msg = int_message()

    verify = Verify(demical_sign, msg, demical_public_exp, demical_modulus)
    return demical sign, verify


def verify site():
    n =
26926241708254230750854269802663410683614241967803392343391714558069148660657297567725587527019155847453318380702729440108740406105001132973539649039020941
    e =
34471067466980120254190492840547446242561760746888287584505511041127927336718233793759838206385995603391623715534044187742326212946080787069642220782687 53

    modulus = 'E65C98C1045DF67E947B6BFE3D1D3F3C0094DCD4073323DB4417F605C0E602EF'
    public_exp = '10001'
    signature = 'D000E36E74573F425D64C57C05B515B7CA428A62ED0DD52068B11BC8FC8D9E0D'

    msg1, demical modulus, demical public exp, hex msg1 = verify site decrypt(public exp,
modulus)

    print(f'\n{modulus = }\n{public_exp = }\n{demical_modulus = }\n{demical_public_exp = }')
    print ()
    print(f'Check decryption:\n{msg1 = }\n{hex_msg1 = }')

    print ()
    n, e, ms2, hex_n, hex_e, hex_msg2 = verify_site_encrypt(e, n)
    print(f'Check encryption:\n{n = }\n{e = }\n\n{hex n = }\n{hex e = }\n{hex msg2 = }')

    print ()
    demical sign, verify = verify site sign(public exp, modulus, signature)
    print(f'Check verification:\n{signature = }\n{demical_sign = }\n{verify = }')
    print ()


if __name__ == '__main__':
    p = 165072043642971037485092457927589701746574998402252816184199736300212458995791
    q = 163118121724427938370395279090619557931144758547815711445975313577132727671651
    p1 = 188875449300751845152394052405139439236103778592153906038817785928665975130443
    q1 = 196346415693867814691019089530626563612968221509038801445814889435600142247623
    Abonent(p, q, p1, q1)
    verify_site()
```

Результат виконання

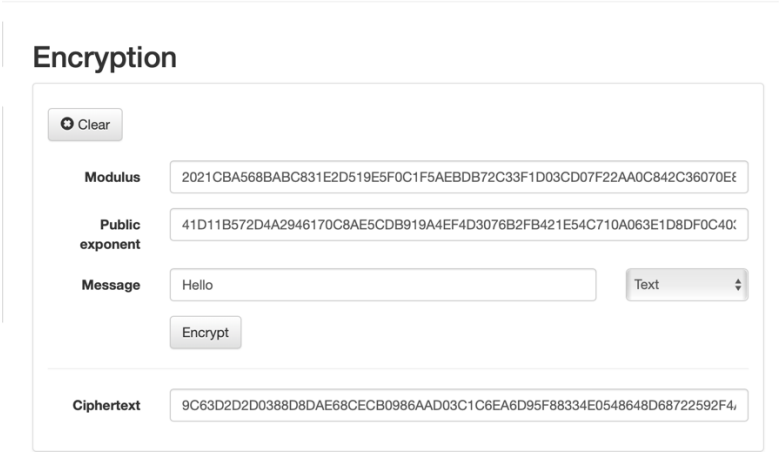
```
+-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------+
Subscriber A:
n = 269262417082542307508542698026634106836142419678033923433917145580691486606572975677255875270191558474533183807027294401087404061050011329735396490390209 41
e = 203646758651058338917033282193292756433055873990478938295100583825852270787560203698292211233854830413838288177697488570509499092617356190503849876436436 3
d = 751836822151136136439541519335799820047069119697363842975647431148781112483159811185872872809766832030908244613982931732864314955426571905371461882230992 7
Subscriber B:
n = 370850174827714768554047532199229721436646179782075915962558160056413025473745624257819015734412596515638173780021966095021186774008838082304940826316869 89
e = 224379963676205785852994732884018455945280594191895609779792466261580037108019531989902635796460765531732039336982356009211383572734756166355432813510999455
d = 134811026493846027284178239592066430416947314400322955124396987076112627656768792110367234083456074859968877132321956260311672203081023150431936552073934 03
+-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------+
k = 1234567890
S = 91186521877366866957297029196933130447707920337062687213762092205942070671465670029030489913351673817657619556829146601163059209528641859050697615631264 87
k1 = 123983757364576237597760514092857411268338675514316120562538377993503771368051741068962567856912146997473662184058153067821167048856762754616573702120308 17
S1 = 357430272455173360382447346562519810723800859665513186958668546873908921330741195536955809350781725802047810940633129960511953386253973873993802356278870 19
+-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------+
k = 1234567890
S = 91186521877366866957297029196933130447707920337062687213762092205942070671465670029030489913351673817657619556829146601163059209528641859050697615631264 87
message = 1234567890
True
+-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------+
+-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------+
Enter your message: ipt kpi
Message in int view: 29678516617048169

Decrypted message:  251590925125077371717135386365778200824779038816142123110808673417079048257128208488452063858738163860899014369428824151602900622568982402908461 9
Encrypted message:  ipt kpi
Signature:  108156922940947015209361349389868777788982359200010704015505667902219206675349368939212175150833449225324913246485513720664395250508120079466934201509362 3
Verify:  True
+-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------+
Enter your message: shovak abdullaieva laba4
Message in int view: 28297947192527444821800363826666281215641372185158097554 44

Decrypted message:  255758580268245416602961850396433294832151944760066908511082435214886652784448820953593960602700608316324693264259225005329142098354906991007049 9
Encrypted message:  shovak abdullaieva laba4
Signature:  219688456570810403571323400339304442763051599995746584095978729368970037524053357932716463114702003800069190878912466246716618389142478346845215485796940 7
Verify:  True
+-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------+
```

Перевірка результатів з сайтом

- шифруємо повідомлення значеннями з програми:
  - результат виконання на сайті

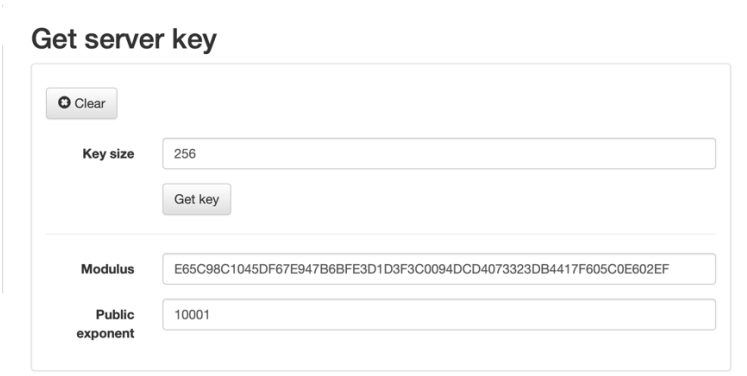


  - перевірка в програмі:

```
+------------------------------------------------------------------------------------------------------------------------------------------------------------------------+
Enter your message: Hello
Check encryption:
n = 269262417082542307508542698026634106836142419678033923433917145580691486606572975677255875270191558474533183807027294401087404061050011329735396490390209 41
e = 34471067466980120254190492840547446242561760746888287584505511041127927336718233793759838206385995603391623715534044187742326212946080787069642220782687 53

hex_n = '2021CBA568BABC831E2D519E5F0C1F5AEBDB72C33F1D03CD07F22AA0C842C36070E86E483DA58A18E0D04268B0448EC0322734A4358B2B74BC1B194F742EAB38D'
hex_e = '41D11B572D4A2946170C8AE5CDB919A4EF4D3076B2FB421E54C710A063E1D8DF0C4030C4D78D2A08CA6D3BB8DF402340FB00E2148FDA852CEFB6EC30485AB551'
hex_msg2 = '9C63D2D2D0388D8DAE68CECB0986AAD03C1C6EA6D95F88334E0548648D68722592F4A4B364D1DC3AEB277974B4D075C1504B18478B23D8BBEA7C9D8253E2C3A5'
+------------------------------------------------------------------------------------------------------------------------------------------------------------------------+
```

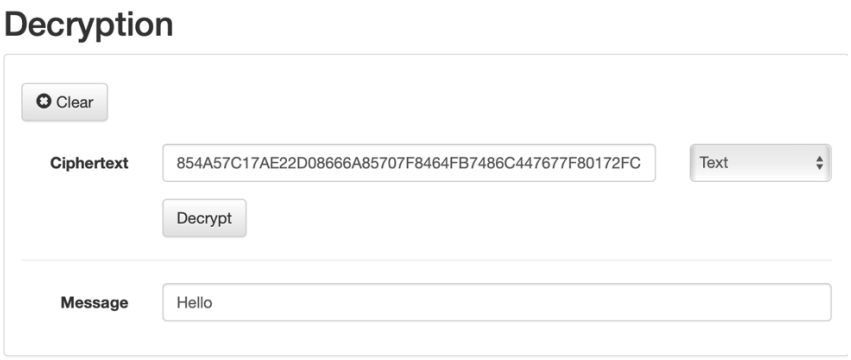
- шифруємо повідомлення значеннями з сайту:



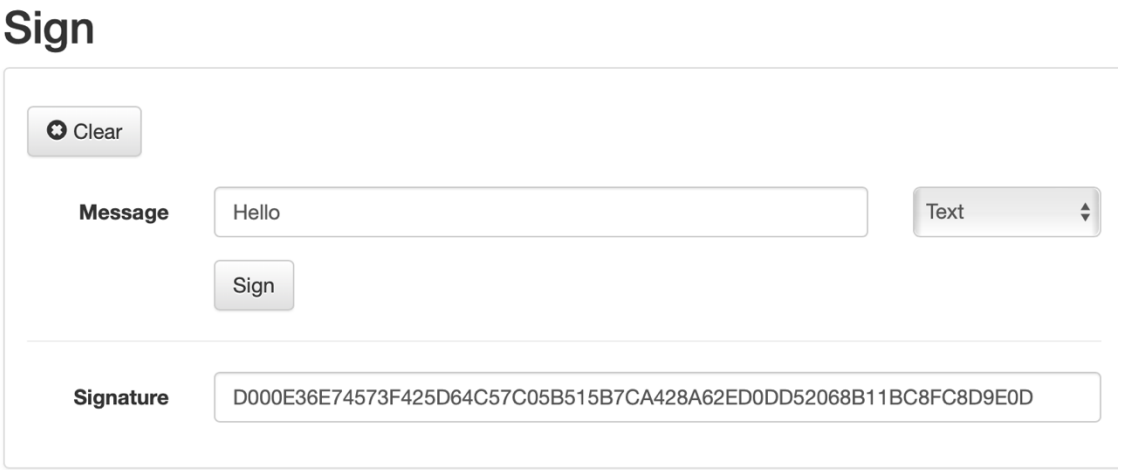
  - зашифровуємо у програмі значеннями з сайту:

```
+------------------------------------------------------------------------------
Check decryption:
msg1 = 310939249775
hex_msg1 = '854A57C17AE22D08666A85707F8464FB7486C447677F80172FC838364F1F373A'
+------------------------------------------------------------------------------
```

- розшифровуємо на сайті:



- перевіряємо підпис значеннями з сайту:
  - створюємо підпис на сайті:



  - перевіряємо в програмі:

```
+----------------------------------------------------------------------------------------------------
Enter your message: Hello
Check verification:
signature = 'D000E36E74573F425D64C57C05B515B7CA428A62ED0DD52068B11BC8FC8D9E0D'
demical_sign = 940826421795863808375514285124431875438827282483485959567816912692197603159 17
verify = True
+----------------------------------------------------------------------------------------------------
```

Кандидати p та q, що не пройшли перевірку на простоту:

```
Failed verification: 203728921079687356573066556672324653534795498561934034518701751413398427900681
Failed verification: 218680701369325908657207177200347228999504576296078982956873674911896906129151
Failed verification: 188492949131622226736877660805110279414789548773626974077536842531948483846967
Failed verification: 217234588547554936329291354155703664195816638535278702728727295450368239323649
Failed verification: 216946861914649161505382863369160018898066448975378371882750676724806378279217
Failed verification: 167583788863288330530684601957024413064311446462281137215956650309233151283553
Failed verification: 179420879021912472564959178967017792385850363133348887056344272983985249760207
Failed verification: 145985478257462560179169059790117698924703757641201467146455643114514835835361
Failed verification: 134445329955680635485483838677989737622702917909683237455172612250031038245163
Failed verification: 213316524493297781446360480754848443190438318265733836517844379126628596242279
Failed verification: 139105280982443625821773280671112065101848947559478812706453495872319738107037
Failed verification: 157736833353517940491079578915436452476671041167060020886570456472158287068233
Failed verification: 152426218407602255838091760082669276705323646454693384976001693134197284944507
Failed verification: 180670975524718564511844791065512104284050772578266112110628114477990538370663
Failed verification: 185101076034028085318291843577746618069662549074396716156530658865749966313433
Failed verification: 117916093610998128444847591713048253070973329504217509874011499954715237847929
Failed verification: 185526791828525796186329884853745417354439779514383124705180463670849820141481
Failed verification: 146027681564745482909093338551711639499893386825619623574301791646034844600871
Failed verification: 139393075879238692447201110287853849229635428837708698629908689992146161268585
Failed verification: 130116908448927800822617108645526630055525383356632129092365199422920882085033
Failed verification: 138731252093078568399721585915659689772952274314154695150789694638703557803517
Failed verification: 209361673807781244822546971871472242035887695482004489103337756253024193654453
Failed verification: 126890448892227196251504116159376879467443135751619998355262494655869258392879
Failed verification: 185793183232202500885757150001093804696720471184069070587204420889828821885529
Failed verification: 195810220828945265665403915009653839315542167273528142692238444417206342098949
Failed verification: 127817783909086353108424264828897133622052346415571095127505376117383582867417
Failed verification: 129691579719785591229109698620761866524321235955608908813787830664688705087363
Failed verification: 229561827219525322855267727775478893686542316552868696678048560229136952602399
Failed verification: 183193165223662007658482948605251283341735629381179273938778107361113063270727
Failed verification: 166878858826306569305842561296593333642396619090012477399456216205291963532047
Failed verification: 167523982907304739597186934751745270103567327840297913730704615255574404646995
Failed verification: 130920425181879695428710759041598333265601973976389001742658467065278728188981
Failed verification: 229094698349840542193810419687283171394811229511046134512343843945306472103069
Failed verification: 184700468215821272725820792412095838576955929161447875660122053796335952565971
Failed verification: 213731428897942387918564201507491873871075892183709154100276570064319360629331
```

Висновок: на цій лабораторній роботі ми ознайомилися з методом Міллера-Рабіна для перевірки простоти великого числа. Розібралися з особливістю роботи системи RSA.