НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»
ФІЗИКО-ТЕХНІЧНИЙ ІНСТИТУТ

КРИПТОГРАФІЯ

КОМП’ЮТЕРНИЙ ПРАКТИКУМ №4

Вивчення криптосистеми RSA та алгоритму електронного підпису.
Ознайомлення з методами генерації параметрів для асиметричних криптосистем.

Підготували студенти групи ФБ-23
Марченко Родіон та Лотиш Андрій

Київ, Жовтень 2024

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 тесту перевірки на простоту рекомендовано використовувати тест Міллера-Рабіна із попередніми пробними діленнями.

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 **p*q <= p1*q1**, де p і q – прості числа для побудови ключів абонента A, а 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 і d1 .

4. Написати програму шифрування, розшифрування і створення повідомлення з цифровим підписом для абонентів A і B.

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. Перевірити роботу програм для випадково обраного ключа 0 < k < n.

1. Напишемо програму, шо реалізує функції криптосистеми RSA мовою python.

Для побудови криптосистеми RSA нам знадобиться знаходити обернений елемент по модулю. Для цього застосуємо функції, написані для лабораторної роботи №3.

`ExtendedEuclidean()` повертає кортеж (двійку), де перший елемент — коефіцієнт Безу для a mod b, а другий — НСД. Обернений елемент — коефіцієнт Безу mod b, якщо НСД=1.

congruencesolver.py:

```
#This module implements the Euclidian algorithm for solving congruences:

#Finds the Bézout coefficient for a mod b and the GCD
def ExtendedEuclidean(a,b):
    old_r, r = a, b
    old_s, s = 1, 0

    while(r!=0):
        q=old_r//r
        old_r, r=r, old_r-q*r
        old_s, s=s, old_s-q*s

    return (old_s%b, old_r) #x, GCD
```

Основна програма:

Функції ASCIItextToNumber() та NumberToASCIItext() перетворюють текст формату ASCII на єдине число і навпаки для подальшого шифрування й дешифрування за схемою RSA. Отримане число відображає строку числових кодів ASCII, де перший символ — найвищий розряд числа у шістнадцятиричному представленні, останній — найнижчий. Наприклад 123 = 0x313233 = 3224115. Даний формат є сумісним з кодуванням, що використовується http://asymcryptwebservice.appspot.com/?section=rsa.

Функція HornerPow(base, exp, mod) підносить надане число base до даного степеня exp за модулем mod за схемою Горнера.
Функція MillerRabin() перевіряє дане число на простоту методом Міллера-Рабіна.
Функція generatePrimeNumber() повертає випадкове просте число заданої довжиини.
Функція GenerateKeyPair() повертає кортеж з усіма необхідними ключами для роботи криптосистеми RSA: ((d, p, q), (e, n)), де d — секретний ключ, e — відкритий ключ (основа степеня), а n — модуль піднесення до степеня.
Функція GenerateKeyPairs() - повертає дві пари відкритих і закритих ключів RSA, де n1 <= n2.
Функції Encrypt() та Decrypt() шифрують і розшифровують числові дані за допомогою наданих відкритого і секретного ключа відповідно.
Функція Sign() додає до повідомлення його цифровий підпис RSA за даним секрктним ключем. Функція Verify() перевіряє цілісність повідомлення за даним підписом RSA та відкритим ключем.
Функції SendKey() та ReceiveKey() імплементують односторонній обмін секретними ключами RSA. SendKey шифрує й підписує повідомлення, а ReceiveKey розшифровує секретне повідомлення і перевіряє його цілісність.

Доступ до функцій забезпечено з допомогою консольного інтерфейсу з параметрами:

```
Custom RSA encryption-decryption & digital certificate program

USAGE:
    -k --key         - Generate new set of two RSA keysets.
    -e --encrypt     - <text> <e> <n> - RSA encrypt.
    -d --decrypt     - <enc> <d> <n> - RSA decrypt.
    -s --sign        - <text> <d> <n> - Sign with RSA digital signature.
    -v --verify      - <text> <signed> <e> <n> - Verify RSA digital signature.
    -sk --sendkey    - <eB> <nB> <dA> <nA> - Send signed message for RSA key exchange.
    -rk --recievekey - <K1> <S1> <dB> <nB> <eA> <nA> - Recieve and process a
                        signed message for RSA key exchange.
    -h --help        - help.
```

Crypto-lab4.1.2.py:

```
#Marchenko Rodion RSA encryption/decryption program ver 1.0:

from congruencesolver import *

import sys
import time
import math, random

BOLD = "\033[1m"
END = "\033[0m"
YELLOW = "\033[0;33m"
BRED = "\033[1;31m"
BGREEN = "\033[1;32m"
```

```
def PrintHelp():
    print("""USAGE:
    -k --key          - Generate new set of two RSA keysets.
    -e --encrypt      - <text> <e> <n> - RSA encrypt.
    -d --decrypt      - <enc> <d> <n> - RSA decrypt.
    -s --sign         - <text> <d> <n> - Sign with RSA digital signature.
    -v --verify       - <text> <signed> <e> <n> - Verify RSA digital signature.
    -sk --sendkey     - <eB> <nB> <dA> <nA> - Send signed message for RSA key exchange.
    -rk --recievekey - <K1> <S1> <dB> <nB> <eA> <nA> - Recieve and process a
                         signed message for RSA key exchange.
    -h --help         - help.
        """)


### DATA PREPROCESSING FUNCTIONS:

#This function encodes a hex representation
# of an ASCII string as a single integral number.
def ASCIItextToNumber(InputText):
    TextBytes = InputText.encode(encoding="ascii")
    Encoded = int.from_bytes(TextBytes, byteorder='big')
    print("  Encrypted data bytes = " + str(hex(Encoded))[2:])
    return Encoded

#This function decodes an integral number to an ASCII string .
def NumberToASCIItext(InputNum):
    print("  Decrypted data bytes = " + str(hex(InputNum))[2:])
    StringLen = int((len(str(hex(InputNum))))/2) - 1
    if (len(str(hex(InputNum))) % 2 == 1):
        StringLen = StringLen + 1
    TextBytes = InputNum.to_bytes(StringLen, 'big')
    Decoded = TextBytes.decode(encoding="ascii")
    return Decoded



### RSA IMPLEMENTATION FUNCTIONS:

#This function raises a number to a given exponent by MOD N
# using the Horner method.
def HornerPow(base, exp, mod):
    lng = exp.bit_length()
    y = 1
    for i in reversed(range(lng)):
        y = (y * y) % mod
        if(not not(exp & (1 << i))):
            y = (y * base) % mod
    return y


#This function implements a prime number searching algorithm.
def MillerRabin(p):
    k = 16
    p1 = p - 1
    bits = format(p1, 'b')
    s = 0
    d = p1
```

```
    while(not(d & 1)):
        s += 1
        d = d >> 1

    for i in range(k):
        x = random.randrange(2, p - 1)
        gcd = ExtendedEuclidean(x, p)[1]
        if(gcd > 1):
            return False
        xr = HornerPow(x, d, p)
        if((xr == 1) or (xr == p1)):
            continue

        for r in range(1, s + 1):
            xr = HornerPow(xr, 2, p)
            if(xr == p1):
                break;
            elif(xr == 1):
                return False
            else:
                continue
        if(xr != p1):
            return False
    return True


#This function returns a random prime number.
def generatePrimeNumber(minlen, maxlen):
    while True:
        p = random.randrange(2**minlen, 2**(maxlen+1))
        if(not (p & 1)):
            continue
        if MillerRabin(p):
            return p
        else:
            continue
```

```
#This function generates a random new RSA keypair.
def GenerateKeyPair():
    p = generatePrimeNumber(256, 512)
    q = generatePrimeNumber(256, 512)
    n = p * q
    e = 65537 #2**16 + 1
    d = ExtendedEuclidean(e, (p-1)*(q-1))[0]
    return ((d, p, q), (e, n))


#This functions creates two key pairs for an RSA exchange,
# where p0q0 <= p1q1.
def GenerateKeyPairs():
    A = GenerateKeyPair()
    B = GenerateKeyPair()
    if((A[0][1]*A[0][2]) > (B[0][1]*B[0][2])):
        A, B = B, A
    return (A, B)



### RSA MAIN FUNCTIONS:

#This function encrypts data using the RSA algorithm.
def Encrypt(InputArray, e, n):
    EncryptedArray = []
    for i in range(0, len(InputArray)):
        Encrypted = HornerPow(InputArray[i], e, n)
        EncryptedArray.append(Encrypted)
    return EncryptedArray
```

```
#This function decrypts data, encrypted using the RSA algorithm.
def Decrypt(InputArray, d, n):
    DecryptedArray = []
    for i in range(0, len(InputArray)):
        Decrypted = HornerPow(InputArray[i], d, n)
        DecryptedArray.append(Decrypted)
    return DecryptedArray


#This function appends data with it`s owner's RSA digital signature.
def Sign(InputArray, d, n):
    SignedArray = []
    for i in range(0, len(InputArray)):
        Signed = HornerPow(InputArray[i], d, n)
        SignedArray.append([InputArray[i], Signed])
    return SignedArray


#This function verifies the RSA signature validity of number data
# using the owner's secret key.
def Verify(InputArray, e, n):
    verifiedArray = []
    for i in range(0, len(InputArray)):
        RecMsg = InputArray[i][0]
        S = InputArray[i][1]
        DecMsg = HornerPow(S, e, n)
        if (DecMsg != RecMsg):
            verifiedArray.append(False)
        else:
            verifiedArray.append(True)
    return verifiedArray



#This function creates a two part message for an RSA key exchange.
def SendKey(e2, n2, d1, n1, K=None):
    #(k2, S2) where k2 = (K**e2)mod n2; S2 = (S1**e2)mod n2; S1 = (K**d1)mod n1
    if (K == None):
        K = generatePrimeNumber(32, 64)

    S1 = Encrypt([K], d1, n1)
    K2 = Encrypt([K], e2, n2)
    S2 = Encrypt([S1[0]], e2, n2)
    return [K, K2[0], S2[0]]


#This function retrieves the secret from an RSA key exchange message
# and verifies it's validity.
def ReceiveKey(Msg, d2, n2, e1, n1):
    # retrieve k = (k2**d2)mod n2; S1 = (S2**d2)mod n2; verify k = (S1**e1)mod n1
    K2 = Msg[0]
    S2 = Msg[1]

    K = Decrypt([K2], d2, n2)
    S1 = Decrypt([S2], d2, n2)
    Ver = Verify([[K[0], S1[0]]], e1, n1)
    return [K[0], Ver[0]]



##### Driver code: #####
argc = len(sys.argv)

print(YELLOW+"""
 Custom RSA encryption-decryption & digital certificate program
    """+END)
```

```
if (argc == 2 and ((sys.argv[1] == "-k") or (sys.argv[1] == "--key"))):
    A, B = GenerateKeyPairs()
    print(BOLD+" A:"+END)
    print("\tPublic Key  (e) = "+str(A[1][0])+"\n\tPrivate key (d) = "+str(A[0][0])
    +"\n\tModulus     (n) = "+str(A[1][1]))

    print(BOLD+"\n B:"+END)
    print("\tPublic Key  (e) = "+str(B[1][0])+"\n\tPrivate key (d) = "+str(B[0][0])
    +"\n\tModulus     (n) = "+str(B[1][1]))

elif (argc == 5):
    if((sys.argv[1] == "-e") or (sys.argv[1] == "--encrypt")):
        print(BOLD+"  RSA Encryption:"+END)
        try:
            Payload = ASCIItextToNumber(sys.argv[2])
            C = Encrypt([Payload], int(sys.argv[3]), int(sys.argv[4]))
            print("\n  Cyphertext     (C) = " + str(C[0]))
            print("  Cyphertext Hex (C) = " + str(hex(C[0]))[2:]+"\n")
        except:
            print(BRED+"\tERROR!"+END+" Cannot encode text data for encryption!")

    elif((sys.argv[1] == "-d") or (sys.argv[1] == "--decrypt")):
        print(BOLD+"  RSA Decryption:"+END)
        try:
            Payload = Decrypt([int(sys.argv[2])], int(sys.argv[3]), int(sys.argv[4]))
            M = NumberToASCIItext(Payload[0])
            print("\n  Cleantext (M) = " + str(M)+"\n")
        except:
            print(BRED+"\tERROR!"+END+" Cannot decode deciphered text data!")


    elif((sys.argv[1] == "-s") or (sys.argv[1] == "--sign")):
        print(BOLD+"  RSA Digital signature:"+END)
        try:
            Payload = ASCIItextToNumber(sys.argv[2])
            Signed = Encrypt([Payload], int(sys.argv[3]), int(sys.argv[4]))
            print("\n  Message       (M) = " + sys.argv[2])
            print("  Signature     (S) = " + str(Signed[0]))
            print("  Signature Hex (S) = " + str(hex(Signed[0]))[2:]+"\n")
        except:
            print(BRED+"\tERROR!"+END+" Cannot sign text data!")

    else:
       PrintHelp()

elif (argc == 6):
    if((sys.argv[1] == "-v") or (sys.argv[1] == "--verify")):
        print(BOLD+"  RSA Digital signature verification:"+END)
        try:
            Payload = ASCIItextToNumber(sys.argv[2])
            Ver = Verify([[Payload, int(sys.argv[3])]], int(sys.argv[4]),
            int(sys.argv[5]))
            if (Ver[0] == True):
                print("\n  Verification == "+BGREEN+"TRUE\n"+END)
            else:
                print("\n  Verification == "+BRED+"FALSE\n"+END)
        except:
            print(BRED+"\tERROR!"+END+" Cannot verify signed text data!")

    elif((sys.argv[1] == "-sk") or (sys.argv[1] == "--sendkey")):
        print(BOLD+"  RSA Key exchange send data:"+END)
        try:
            K, K2, S = SendKey(int(sys.argv[2]), int(sys.argv[3]), int(sys.argv[4]),
            int(sys.argv[5]), K=None)
            print("\n  Secret     (K) = " + str(K))
            print("  Key       (K1) = " + str(K2))
            print("  Signature (S1) = " + str(S)+"\n")

        except:
            print(BRED+"\tERROR!"+END+" Cannot create RSA key exchange message!")

    else:
       PrintHelp()
```

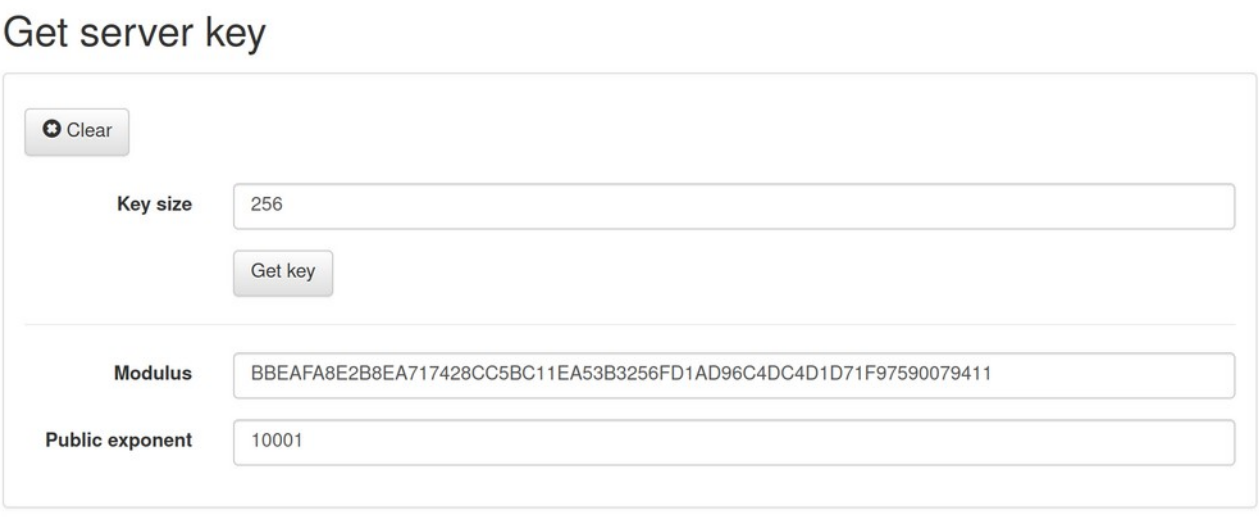
```
elif(argc == 8 and ((sys.argv[1] == "-rk") or (sys.argv[1] == "--recievekey"))):
    print(BOLD+"  RSA Key exchange recieve data:"+END)
    try:
        K, Ver = ReceiveKey([int(sys.argv[2]), int(sys.argv[3])], int(sys.argv[4]),
        int(sys.argv[5]), int(sys.argv[6]), int(sys.argv[7]))
        print("\n  Key decrypted (K) = " + str(K))
        if (Ver == True):
            print("  Verification == "+BGREEN+"TRUE\n"+END)
        else:
            print("  Verification == "+BRED+"FALSE\n"+END)

    except:
        print(BRED+"\tERROR!"+END+" Cannot decode recieved RSA key exchange pair!")

else:
    PrintHelp()
```

2. Демонстрація роботи.

Для перевірки правильності побудови криптосистеми скористаємося сумісною онлайн версією на сайті http://asymcryptwebservice.appspot.com/?section=rsa.
Для переводу формату чисел між шістнадцятковою і десятковою системами й навпаки використовуватимемо сайт https://www.rapidtables.com/convert/number/hex-to-decimal.html.

1. Генерація ключів. Згенеруємо набори відкритих і закритих ключів RSA локально і на сайті:

```
rodion@rodion-Vivobook:~/CRYPT-GIT/lab4-files$ python3 Crypto-lab4.1.py -k

 Custom RSA encryption-decryption & digital certificate program

 A:
       Public Key  (e) = 65537
       Private key (d) = 174508570391776340820751243546370868177283305746542908999203654641048579487480164058731679526685594
071544370864223397193330387106585898742661169675382590690090019477187893616581022946702196527079346985813998855243389979728645757065004654542390445930422554626766442106684873210482505893231346601501682769184 1
       Modulus     (n) = 956251519880087462238258716412918694626640134507623965474984106539331166711621029424506528523444295875150788739850232596930901321431441964697748914343549222183483222158012749517621364605538175001120882726072363532796101176644612890366245211524692449141320445770868193929173531362696313072783800515161408677 47

 B:
       Public Key  (e) = 65537
       Private key (d) = 38635491911764521412944436197109138657542785838425940393943901353281138577689920496539365343765365432750674051723475700600516415393071043836725573106425927585260612691131702729275543305422097918563379679710255139984496089076038010290971807261449411965621680837695964601368415453616419167176723865064364269549
       Modulus     (n) = 14883055507090527478047020014400409217653450628889207403737738570446047016787540819265857794241760738060194705976614512374396310560252142478854310731063518670777855979010195613247336293547018115092324642168490666838725921675878766699348873087812058204020656420482194849290934504208278651005788619143149365726 1
```

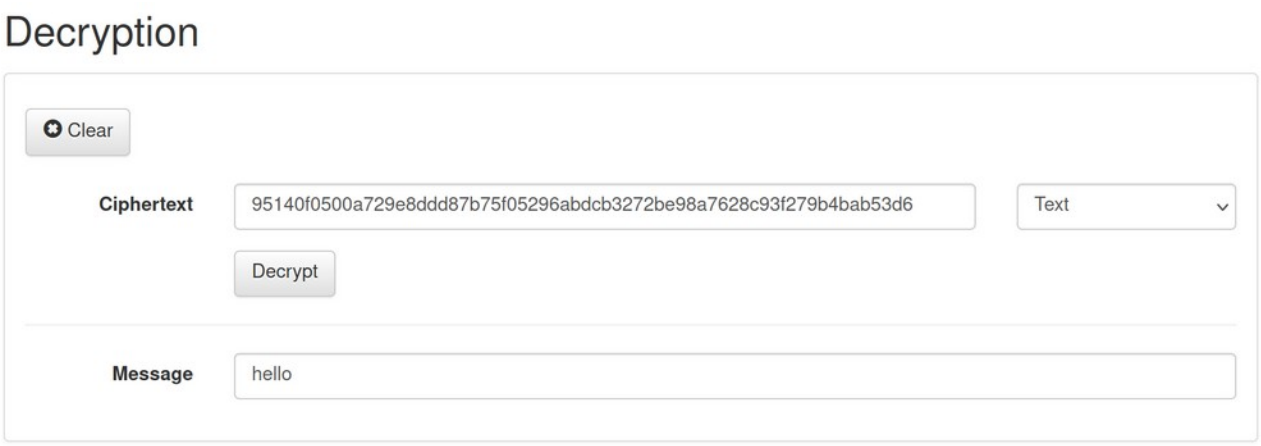


Переведемо ключ сайту в десятковий формат:

```
Keys from site:

Modulus     (n) = BBEAFA8E2B8EA717428CC5BC11EA53B3256FD1AD96C4DC4D1D71F97590079411.
decimal(BBEAFA8E2B8EA717428CC5BC11EA53B3256FD1AD96C4DC4D1D71F97590079411) =
84997674167715193393463210718649314768822069971318543708344448234114989397009

Public Key  (e) = 10001.
decimal(10001) = 65537
```

2. Шифрування. Зашифруємо текстове повідомлення локально з допомогою відкритого ключа сайту й розшифруємо на сервері:

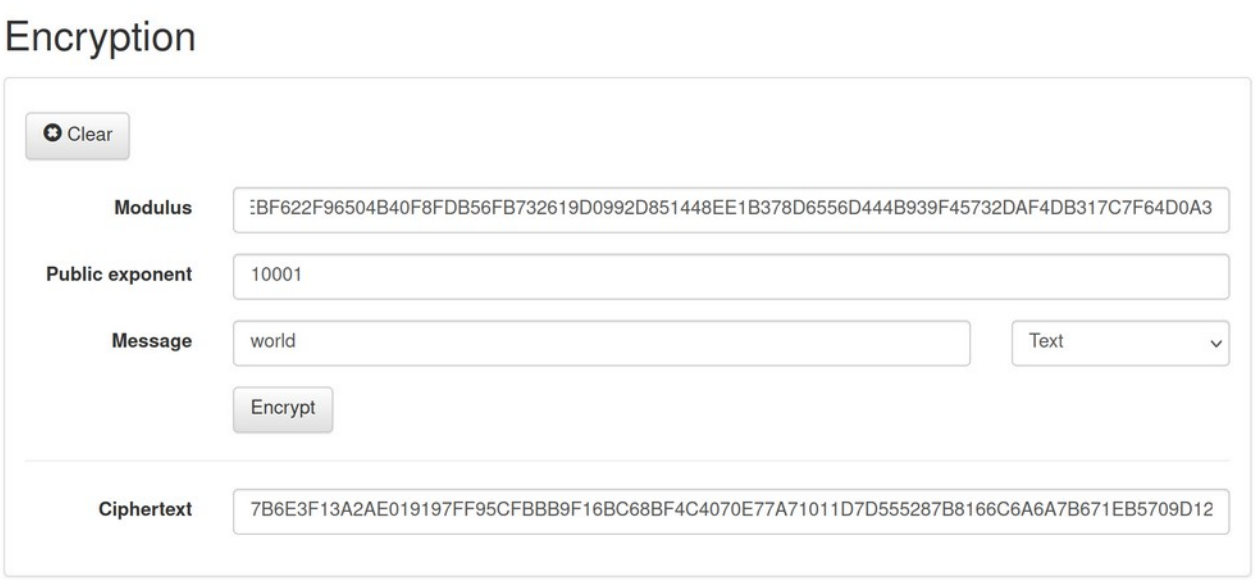
```
rodion@rodion-Vivobook:~/CRYPT-GIT/lab4-files$ python3 Crypto-lab4.1.py -e hello 65537 84997674167715193393463210718649314768
822069971318543708344448234114989397009

 Custom RSA encryption-decryption & digital certificate program

 RSA Encryption:
 Encrypted data bytes = 68656c6c6f

 Cyphertext     (C) = 67430055041265961909882193008031676139598955610073385059240939443059732206550
 Cyphertext Hex (C) = 95140f0500a729e8ddd87b75f05296abdcb3272be98a7628c93f279b4bab53d6
```



3.Дешифрування. Зашифруємо повідомлення на сервері з допомогою локального відкритого ключа A та розшифруємо локально з допомогою секретного ключа:

Переведемо локальні ключі в шістнадцяткову систему числення:

```
A:
hex(65537) = 10001
hex(n) =
882CBB895324F25E72C535DEB7C5796C18AF08B36FB334974BBD9E1B3C66AAB1FD66325CDF52238CA5C0C72
E2C3A265033DB240EF4AC568ABDC78FC1E88ACDEF1C52A0B4DD87F307A3A729C589D60B68D5D9E04637EBF6
22F96504B40F8FDB56FB732619D0992D851448EE1B378D6556D444B939F45732DAF4DB317C7F64D0A3
```



Переведемо шифроване повідомлення у весяткову систему:

```
M =
7B6E3F13A2AE019197FF95CFBBB9F16BC68BF4C4070E77A71011D7D555287B8166C6A6A7B671EB5709D1242
A3500890ED002ED6316CF630FDD7D28F74208862B04AF139D054450E37248C170CF059AD6286F86CD57B94B
0A277FD85E74B2753369453714C1912E941427E1AF6C27A2DDD80B53DDE66DC3421306B2D7965237D9

decimal(M) =
866759500351523181347701898994510705988554713224848446872965276837130054976171447190828
376755096917840467072717275029476402945698644085902923745924227367302464398171043154758
059474411550129972969242428694187138847351246762963731747876902729710053433483880387738862717431437068514312868538168783461352
67325913 174508570391776340820751243546370868177283305746542908999203654641048579487480164058731679526685594071544370864223397193330387106585898742661169675382590690090019477187893616581022946702196527079346985813998855243389979728645757065004654542390445930422554626766442106684873210482505893231346601501682769184 956251519880087462238258716412918694626640134507623965474984106539331166711621029424506528523444295875150788739850232596930901321431441964697748914343549222183483222158012749517621364605538175001120882726072363532796101176644612890366245211524692449141320445770868193929173531362696313072783800515161408677477
```

```
Custom RSA encryption-decryption & digital certificate program

RSA Decryption:
Decrypted data bytes = 776f726c64

Cleantext (M) = world
```

4. Цифровий підпис. Підпишемо повідомлення локально з допомогою секретного ключа A та перевіримо автентичність на сервері з допомогою локального відкритого ключа:

Переводимо формат локальних ключів:

```
A:
hex(65537) = 10001
hex(n) =
882CBB895324F25E72C535DEB7C5796C18AF08B36FB334974BBD9E1B3C66AAB1FD66325CDF52238CA5C0C72
E2C3A265033DB240EF4AC568ABDC78FC1E88ACDEF1C52A0B4DD87F307A3A729C589D60B68D5D9E04637EBF6
22F96504B40F8FDB56FB732619D0992D851448EE1B378D6556D444B939F45732DAF4DB317C7F64D0A3
```

```
rodion@rodion-Vivobook:~/CRYPT-GIT/lab4-files$ python3 Crypto-lab4.1.py -s crypto4 17450857039177634082075124354637086817728330574654290899920365464104857948748016405873167952668559407154437086422339719333038710658589874266116967538259069009001947718789361658102294670219652707934698581399885524338997972864575706500465454239044593042255462676644210668487321048250589323134660150168276918419562515198800874622382587164129186946266401345076239654749841065393311667116210294245065285234442958751507887398502325969309013214314419646977489143435492221834832221580127495176213646055381750011208827260723635327961011766446128903662452115246924449141320445770868193929173531362696313072783800515161408677470x
```

Custom RSA encryption-decryption & digital certificate program

RSA Digital signature:
Encrypted data bytes = 63727970746f34

Message (M) = crypto4
Signature (S) = 272295439694899645842522105182034807317585496747888365889409572866756509721808561647773092364501822612750893964127859339816445624471041672266761768541569286458547857076294055751626432996186939174696401723690390723440803778991486461167022974853525028961140438803578607027009393536879287232637016357121671678673457

Signature Hex (S) = 26c6b2460522d766ecc78383e71874b6f104e324cce3d5a94136816b4998e29f8f290f9ab65f017dbacf867230a31231a1c679e94520c9ff2790a9a699257ca3dac5b0079fd5406a3b4e1869498f6a19cfcb144afd7e8a3088e4dfde41c9727954902bec26d3c97530555e289d7df319c2d4298e53fdbacda6a75fa5a71a73d1

Verify

Clear

Message crypto4 — Text

Signature cb144afd7e8a3088e4dfde41c9727954902bec26d3c97530555e289d7df319c2d4298e53fdbacda6a75fa5a71a73d1

Modulus BF622F96504B40F8FDB56FB732619D0992D851448EE1B378D6556D444B939F45732DAF4DB317C7F64D0A3

Public exponent 10001

Verify

Verification true ✔

5. Перевірка автентичності. Підпишемо повідомлення на сервері цифровим підписом та перевіримо його автентичність локально, застосовуючи відкриті ключі сайту:

Sign

Clear

Message hello world! — Text

Sign

Signature 30BE5DE0C382981215128E9B6881E4A5405B272326487CD6C27670CA13675962

Переведемо підпис у десяткову систему:

```
M = hello world!
S = 30BE5DE0C382981215128E9B6881E4A5405B272326487CD6C27670CA13675962
decimal(S) =
22047365596332604770284776772774215095067295813308789745604334773024116398434
```

```
rodion@rodion-Vivobook:~/CRYPT-GIT/lab4-files$ python3 Crypto-lab4.1.py -v hello\ world! 22047365596332604770284776772774215095067295813308789745604334773024116398434 65537 84997674167715193393463210718649314768822069971318543708344448234114989397009

 Custom RSA encryption-decryption & digital certificate program

  RSA Digital signature verification:
  Encrypted data bytes = 68656c6c6f20776f726c6421

  Verification == TRUE
```

Перевірка успішна!

```
rodion@rodion-Vivobook:~/CRYPT-GIT/lab4-files$ python3 Crypto-lab4.1.py -v hello\ world! 22047365596332604770284776772774215095067295813308789745604334773024116398434 65537 95625151988008746223825871641291869462664013450762396547498410653933116671162102942450652852344429587515078873985023259693090132143144196469774891434354922218348322215801274951762136460553817500112088272607236353279610117664461289036624521152469244914132044577086819392917353136269631307278380051516140867747

 Custom RSA encryption-decryption & digital certificate program

  RSA Digital signature verification:
  Encrypted data bytes = 68656c6c6f20776f726c6421

  Verification == FALSE
```

При подачі ключа, що не відповідає даному підпису або навпаки, перевірка виводить невідповідність (False).

6. Відправлення шифрованого повідомлення обміну ключами RSA. Оскільки ми не можемо гарантувати, що модуль відкритого ключа сервера більший за модуль відкритого ключа локальної криптосистеми, то дану перевірку виконаємо локально застосовуючи попередньо згенеровані пари ключів A та B.

```
rodion@rodion-Vivobook:~/CRYPT-GIT/lab4-files$ python3 Crypto-lab4.1.py -sk 65537 14883055507090527478047020014400409217653450628889207403737738570446047016787540819265857794241760738060194705976614512374396310560252142478854310731063518670777855979010195613247336293547018115092324642168490666838725921675878766699348873087812058204020656420482194849290934504208278651005788619143149365726117450857039177634082075124354637086817728330574654290899920365464104857948748016405873167952668559407154437086422339719333038710658589874266116967538259069009001947718789361658102294670219652707934698581399885524338997972864575706500465454239044593042255462676644210668487321048250589323134660150168276918419562515198800874622382587164129186946266401345076239654749841065393311667116210294245065285234442958751507887398502325969309013214314419646977489143435492221834832221580127495176213646055381750011208827260723635327961011766446128903662452115246924491413204457708681939291735313626963130727838005151614086774739562515198800874622382587164129186946266401345076239654749841065393311667116210294245065285234442958751507887398502325969309013214314419646977489143435492221834832221580127495176213646055381750011208827260723635327961011766446128903662452115246924491413204457708681939291735313626963130727838005151614086774739

 Custom RSA encryption-decryption & digital certificate program

  RSA Key exchange send data:

  Secret     (K) = 29821461502565245517
  Key       (K1) = 13769409761835675848098531753738294759004245078090899120840170593187238772679859384745777162698824846863240299326455045044511184524932727909138676905524600212201457059087774575686719490024664713037058934775892234628211591582512100012690012475002615304761771369138832015201166849414214249978184878889071245619 6
  Signature (S1) = 52265971127098995839344010692658209891196366441670323493585562114503585375582177795777204382577929953817859584429196251377527806386539527318848612113102007783352137128677625805984686127351842054736990836982198541133409230299257890491163885914392759096145643442395255117998022167451657153044336061364820118202
```

Повідомлення з випадковим числом K клієнта A шифруємо відкритим ключем B.

```
rodion@rodion-Vivobook:~/CRYPT-GIT/lab4-files$ python3 Crypto-lab4.1.py -rk 13769409761835675848098531753738294759004245078090899120840170593187238772679859384745777162698824846863240299326455045044511184524932727909138676905524600212201457059087774575686719490024664713037058934775892234628211591582512100012690012475002615304761771369138832015201166849414214249978184878889071245619662265971127098995839344010692658209891196366441670323493585562114503585375582177957772043825779299538178595844291962513775278063865395273188486121131020077833521371286776258059846861273518420547369908369821985411334092302992578904911638859143927590961456434423952551179980221674516571530443360613648201182023863549191176452141294443619710913865754278583842594039394390135328113857768992049653936534376536543275067405172347570060051641539307104383672557310642592758526061269113170272927554330542209791856337967971025513998449608907603801029097180726144941196562168083769596460136841545361641916717672386506436426954916488305550709052747804702001440040921765345062888920740373773857044604701678754081926585779424176073806019470597661451237439631056025214247885431073106351867077785597901019561324733629354701811509232464216849066683872592167587876669934887308781205820402065642048219484929093450420827865100578861914314936572616549841065393311667116210294245065285234442958751507887398502325969309013214314419646977489143435492221834832221580127495176213646055381750011208827260723635327961011766446128903662452115246924491413204457708681939291735313626963130727838005151614086774748830555070905274780470200144004092176534506288892074037377385704460470167875408192658577942417607380601947059766145123743963105602521424788543107310635186707778559790101956132473362935470181150923246421684906668387259216758787666993488730878120582040206564204821948492909345042082786510057886191431493657261 65537 9562515198800874622382587164129186946266401345076239654749841065393311667116210294245065285234442958751507887398502325969309013214314419646977489143435492221834832221580127495176213646055381750011208827260723635327961011766446128903662452115246924491413204457708681939291735313626963130727838005151614086774748
```

```
Custom RSA encryption-decryption & digital certificate program

RSA Key exchange recieve data:

Key decrypted (K) = 29821461502565245517
Verification == TRUE
```

Розшифровуємо й перевіряємо цілісність повідомлення від клієнта A з допомогою секретного ключа B. Отримали співпадіння і розшифроване значення K.

```
rodion@rodion-Vivobook:~/CRYPT-GIT/lab4-files$ python3 Crypto-lab4.1.py -rk 13769409761835675848098531753738294759004245078090899120840170593187238772679859384745777162698824846863240299326455045044511184524932727909138676905524600212201457059087774575686719490024664713037058934775892234628211591582512100012690012475002615304761771369138832015201166849414214249978184878889071245619662265971127098995839344010692658209891196366441670323493585562114503585375582177957772043825779299538178595844291962513775278063865395273188486121131020077833521371286776258059846861273518420547369908369821985411334092302992578904911638859143927590961456434423952551179980221674516571530443360613648201182023863549191176452141294443619710913865754278583842594039394390135328113857768992049653936534376536543275067405172347570060051641539307104383672557310642592758526061269113170272927554330542209791856337967971025513998449608907603801029097180726144941196562168083769596460136841545361641916717672386506436426954916488305550709052747804702001440040921765345062888920740373773857044604701678754081926585779424176073806019470597661451237439631056025214247885431073106351867077785597901019561324733629354701811509232464216849066683872592167587876669934887308781205820402065642048219484929093450420827865100578861914314936572616549841065393311667116210294245065285234442958751507887398502325969309013214314419646977489143435492221834832221580127495176213646055381750011208827260723635327961011766446128903662452115246924491413204457708681939291735313626963130727838005151614086774748
```

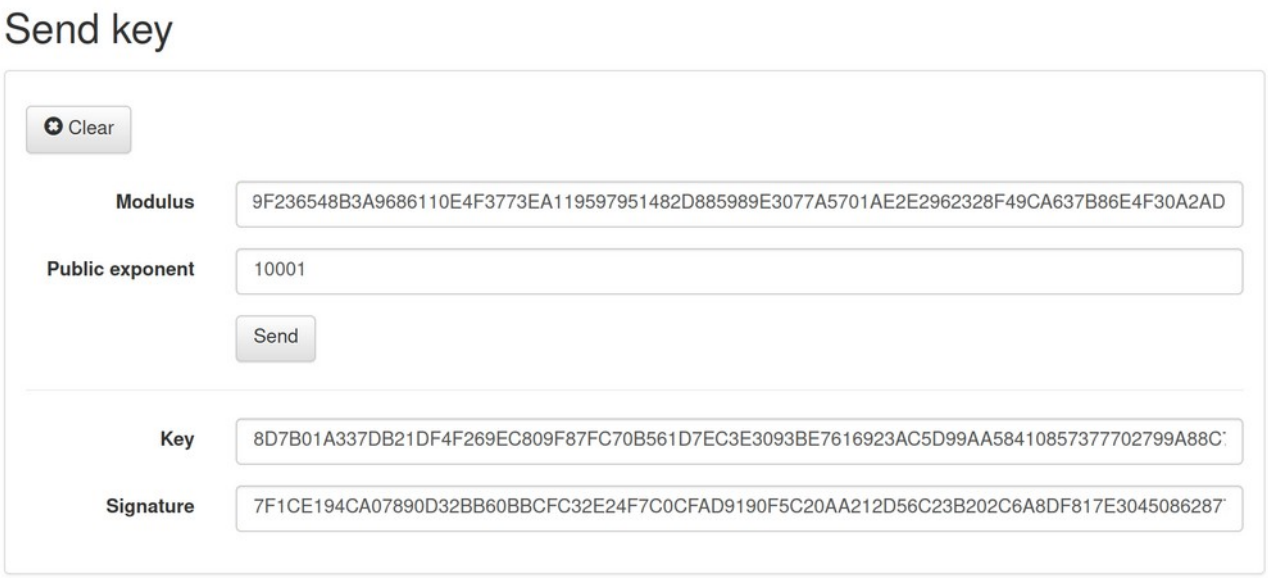
```
Custom RSA encryption-decryption & digital certificate program

RSA Key exchange recieve data:

Key decrypted (K) = 29821461502565245517
Verification == FALSE
```

При зміні відкритого ключа A відносно шифрованого повідомлення й навпаки викликає помилку верифікації.

7. Отримання й дешифрування повідомлення обміну ключами RSA. Предамо локальному клієнту шифроване повідомлення з секретним значенням K з сервера, використовуючи для передачі відкриті локальні ключі A:

Переведемо ключі в шістнадцяткову систему:

```
A = Website. B = My script:

B:
hex(65537) = 10001
hex(n) =
D3F1163CA78C0A5C40B978D4F6CB58A12AB3B3F957C85E7F6D17DF298881015362B83BC95254E5365A91AAE
699E1590BA566492F834CEDA93FCD02C630B076708E8EBB4F37884E384032C86DB7A8677C77D68FC5E9F236
548B3A9686110E4F3773EA119597951482D885989E3077A5701AE2E2962328F49CA637B86E4F30A2AD
```



Створили повідомлення.

Переводимо шифровані дані в десяткову систему:

```
A = Website. B = My script:

K1 =
8D7B01A337DB21DF4F269EC809F87FC70B561D7EC3E3093BE7616923AC5D99AA58410857377702799A88C78
260732CB06190AB13962BF85031FCFD7D355AC16C2039A53DC5839F8D8A7DEB00DEE39377D0B745CCF60B1C
58B8048639AB1620F4AFC79AFFE8B6A4403EB7C812F5E851F9883C60F5945CF83CFF643903C73C9B59
decimal(K1) =
99350981370533997983070750114289655526797199829737711087567977923994355760396458713394
32545994023852326918148217458785449700156349689677166459111935169350797786460945011725960
15219675422487341750959050762825569581787090652140454215969996195156241706179000251453
47897566981920109126357406364262880490225965913

S1 =
7F1CE194CA07890D32BB60BBCFC32E24F7C0CFAD9190F5C20AA212D56C23B202C6A8DF817E30450862877C9
FB7797E0CC47E036E1BC36EFB31920E6F5F9BFE64C4C98AA52D8E67C1B91632E50B0DCCDE52D0E256EB5DD6
E90690184778D7EF3D690D4460D15398F7408FB3153B083CFFE426BF8948464C37295EB3A40C21C307
decimal(S1) =
89261655721330277979951876674044303895594469264916646592475113914937956870782674645321930982534554186784891548048744
13250069662622936402197683966925450991324411860397537920304451823872743055950967613364096624902254874990203862867374518865440
04256561823465073446386093580360280296626757952801235838419526075143 38635491911764521412944436197109138657542785838425940393
94390135328113857768992049653936534376536543275067405172347570060051641539307104383672557310642592758526061269113170272927554
33054220979185633796797102551399844960890760380102909718072614494119656216808376959646013684154536164191671767238650643642695
49 14883055070905274780470200144004092176534506288892074037377385704460470167875408192658577942417607380601947059766145123743
96310560252142478854310731063518670777855979010195613247336293547018115092324642168490666838725921675878766699348873087812058
20402065642048219484929093450420827865100578861914314936572610 65537 84997674167715193393463210718649314768822069971318543708
344448234114989397009

Custom RSA encryption-decryption & digital certificate program

RSA Key exchange recieve data:

Key decrypted (K) = 16916474917802015318
Verification == TRUE
```

Як видно, повідомлення цілісне і секретне значення передано.

Лістинг роботи з локальною криптосистемою в консолі наданий у файлі “Rsa-script-usage.log”.
Випадкові ключі даного сеансу надані у файлі “Demo-keys-used.txt”

Висновки:

В цій роботі ми ознайомилися з основами побудови базових асиметричних криптосистем на прикладі реалізації криптосистеми RSA, з основами цифрового підпису та обміну секретними даними. Було створено власну криптосистему, що може шифрувати і дешифрувати дані, підписувати цифровим підписом та обмінюватися секретними ключами по відкритому каналу за схемами RSA. В процесі побудови криптосистеми ми також отримали базові знання у сфері реалізації алгоритмів модульної арифметики на прикладі схеми Горнера для піднесення в степінь та пошуку простих чисел за модулем.