Лабораторна робота 2 з "Асиметричних криптосистем та протоколів"

Тема: Вивчення криптосистеми RSA та алгоритму електронного підпису

Виконали
Бондар Петро, ФІ-03
Кістаєв Матвій, ФІ-03

In [ ]:
```
import numpy as np
import random
import math
import hashlib
import requests
```

Генератор простих чисел

Для генерації чисел ми використаємо генератор BBS з лабораторної роботи 1.

Для узагальнення генерації ми зберігатимемо стан генератора, на якому зупинилась попередня послідовність.

In [ ]:
```
class BBS:
    def __init__(self, p, q, state = 0):
        self.n = p*q
        self.state = state

        if state == 0:
            self.state = random.randint(2, self.n - 1)

    def generate_bytes(self, n: int):
        seq = np.zeros(n, dtype=object)
        seq[0] = self.state

        for i in range(1, n):
            seq[i] = pow(seq[i - 1], 2, self.n)

        self.state = seq[-1]
        seq = np.array(seq % (2**8), dtype=np.uint8)

        return seq

def bytes_to_num(byte_seq):
    res = 0
    for b in byte_seq:
        res = res*(2**8) + int(b)

    return res
```

Після чого, згенеровані числа заданої довжини будуть перевірятися на сильну простоту:

1. Спочатку саме згенероване число n перевіряється на простоту.

2. Після чого на простоту перевіряється число $\frac{n-1}{2}$.

Для перевірки числа на простоту ми скористаємося комбінацією методу пробних ділень та алгоритмом Міллера-Рабіна.

In [ ]:
```
OPTIMUS_PRIMES = [2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59,
                  61, 67, 71, 73, 79, 83, 89, 97, 101, 103, 107, 109, 113, 127, 131,
                  137, 139, 149, 151, 157]
R = {}

for d in OPTIMUS_PRIMES:
    R[d] = [1]
    while R[d].count(R[d][-1]) < 2:
        R[d].append((R[d][-1] * 2) % d)
    R[d].pop()

# Метод пробних ділень
def petod_drobnyx_mylen(num):
    b = bin(num)[:1:-1]

    if b[0] == '0':
        return 2

    for d in OPTIMUS_PRIMES[1::]:
        sum = 0
        for i in range(len(b)):
            sum += int(b[i]) * R[d][i % len(R[d])]
            sum %= d

        if sum == 0:
            return d

    return 1

# Ймовірнісний алгоритм Міллера-Рабіна та загальний алгоритм для знаходження простих чисел
def miller_rabin(num, base):
    i = 1
    while (num - 1) % (2 ** i) == 0:
        i += 1

    k = i - 1
    d = (num - 1) // (2 ** k)

    a_d = pow(base, d, num)

    if a_d == 1:
        return True

    a_d2i = a_d
    for j in range(k):
        if a_d2i == (num - 1):
            return True

        a_d2i = (a_d2i ** 2) % num

    return False
```

```python
def check_prime(num, error_prob = 0.001):
    if petod_drobnyx_mylen(num) != 1:
        return False

    t = int(math.ceil(math.log(1 / error_prob, 4)))
    s = 0
    for _ in range(t):
        a = random.randrange(3, num + 1)
        s += int(miller_rabin(num, a))

    return s > (t / 2)

# Генератор простих чисел
def generate_prime(len: int, excl = []):
    gen = BBS(int('425D2B9BFDB25B9CF6C416CC6E37B59C1F', 16), int('D5BBB96D30086EC484EBA3D7F9CAEB07', 16))

    while True:
        p = bytes_to_num(gen.generate_bytes(len // 8))
        if check_prime(p) and (p not in excl):
            return p

# Генератор сильнопростих чисел
def generate_safe_prime(len: int, excl = []):
    gen = BBS(int('425D2B9BFDB25B9CF6C416CC6E37B59C1F', 16), int('D5BBB96D30086EC484EBA3D7F9CAEB07', 16))

    while True:
        seq = gen.generate_bytes(len // 8)
        if seq[0] < 128:
            continue

        p = bytes_to_num(seq)
        if not check_prime(p) or (p in excl):
            continue

        q = (p - 1) // 2
        if check_prime(q):
            return p
```

Взаємодія з віддаленим сервером

У якості сервера з яким ми будемо спілкуватися ми скористалися **asymcryptwebservice**, який нам люб'язно надав Олег Миколайович.

Клас **Server** дозволяє налагодити сесію спілкування з сервісом та надає зручний інтерфейс для надсилання запитів, що необхідні для цієї лабораторної роботи.

In [ ]:
```python
KEY_LENGTH = 256


class Server:
    __base_url = 'http://asymcryptwebservice.appspot.com/rsa/'
    s = requests.Session()
    n = None
    e = None

    # Setup server private key and receive server pub key
    def set_server_key(self, key_l: int) -> (str, str):
        req = f'{self.__base_url}serverKey?keySize={key_l}'
```

```
        print(f"H: Sent request: {req}")
        r = self.s.get(req)
        if r.status_code != 200:
            raise RuntimeError(f"Incorrect server status code {r.status_code}:\n\tRequest {req}\n\tResponse{r.json()}")
        r = r.json()
        print(f"S: Response: {r}")
        self.n, self.e = (int(r['modulus'], 16), int(r['publicExponent'], 16))
        return (self.n, self.e)

    # Ask server to encrypt
    def encrypt(self, M: str, rec_n, rec_e, type='TEXT'):
        req = f'{self.__base_url}encrypt?modulus={format(rec_n, "X")}&publicExponent={format(rec_e, "X")}&message={M}&type={type}'
        print(f"H: Sent request: {req}")
        r = self.s.get(req)
        if r.status_code != 200:
            raise RuntimeError(f"Incorrect server status code {r.status_code}:\n\tRequest {req}\n\tResponse{r.json()}")
        r = r.json()
        print(f"S: Response: {r}")
        return r['cipherText']

    # Ask server to decrypt this message with his private keys
    def decrypt(self, C: str, type='TEXT'):
        req = f'{self.__base_url}decrypt?cipherText={C}&expectedType={type}'
        print(f"H: Sent request: {req}")
        r = self.s.get(req)
        if r.status_code != 200:
            raise RuntimeError(f"Incorrect server status code {r.status_code}:\n\tRequest {req}\n\tResponse{r.json()}")
        r = r.json()
        print(f"S: Response: {r}")
        return r['message']

    # Ask server to sign this message with his private keys
    def sign(self, M: str, type='TEXT'):
        req = f'{self.__base_url}sign?message={M}&type={type}'
        print(f"H: Sent request: {req}")
        r = self.s.get(req)
        if r.status_code != 200:
            raise RuntimeError(f"Incorrect server status code {r.status_code}:\n\tRequest {req}\n\tResponse{r.json()}")
        r = r.json()
        print(f"S: Response: {r}")
        return r['signature']

    # Verify the message using this public key
    def verify(self, M: str, sign: str, u_n, u_e, type='TEXT'):
        req = f'{self.__base_url}verify?message={M}&type={type}&signature={sign}&modulus={format(u_n, "X")}&publicExponent={format(u_e, "X")}'
        print(f"H: Sent request: {req}")
        r = self.s.get(req)
        if r.status_code != 200:
            raise RuntimeError(f"Incorrect server status code {r.status_code}:\n\tRequest {req}\n\tResponse{r.json()}")
        r = r.json()
        print(f"S: Response: {r}")
        return r['verified']

    # Receive a pair (64bit encrypted key, signature for this key) from the server
    def sendKey(self, rec_n, rec_e) -> (str, str):
        req = f'{self.__base_url}sendKey?modulus={format(rec_n, "X")}&publicExponent={format(rec_e, "X")}'
        print(f"Sent request: {req}")
        r = self.s.get(req)
```

```
        if r.status_code != 200:
            raise RuntimeError(f"Incorrect server status code {r.status_code} for request {req}")
        r = r.json()
        print(f"response: {r}")
        return (r["key"], r['signature'])

    # Ask server to decrypt and verify key encrypted with user's modulo and publicExponent
    def receiveKey(self, K_enc: str, sign, u_n, u_e):
        req = f'{self.__base_url}receiveKey?key={K_enc}&signature={sign}&modulus={format(u_n, "X")}&publicExponent={format(u_e, "X")}'
        print(f"H: Sent request: {req}")
        r = self.s.get(req)
        if r.status_code != 200:
            raise RuntimeError(f"Incorrect server status code {r.status_code}:\n\tRequest {req}\n\tResponse{r.json()}")
        r = r.json()
        print(f"S: Response: {r}")
        return r['verified']
```

Опис процесу спілкування

Службові функції перетворення

Іноді дані необхідно відправити у певному форматі, а бо вони приходять у цьому форматі.

Для перетворення даних між різними формами довелося імплементувати методи **str2hex** та **num2str**.

In [ ]:
```
def str2hex(s: str):
    res = ""

    for c in s:
        cb = hex(ord(c))
        res += cb[2::]

    return res

def num2str(n: int):
    text = str()
    while n != 0:
        text += chr(n % 256)
        n //= 256

    return text[::-1]
```

Абстракція користувача

Для спілкування необхідний абонент, що буде надсилати, отримувати та оброблювати запити від сервера. Для цього створений клас **User**:

1. При ініціалізації він приймає значення **p** та **q**, що будуть його публічним ключем, а також опціонально публічну експоненту **e** та сервер. У випадку відсутності заданого сервера створиться новий об'єт та запуститься нова сесія спілкування з сервером. В будь-якому випадку, при створенні коритсувача будуть заново згенеровані публічний та особистий ключ на стороні сервера, що буде використовуватись під час взаємодіїю
2. Метод **send_message** дозволяє зашифрувати та надіслати повідомлення **M** серверу відповідно до протоколу спілкування. У відповідь користувач повинен розшифроване сервером повідомлення.
3. Метод **send_message_sign** дозволяє отримати підпис **S** для повідомлення **M** та надіслати цю пару серверу напряму для верифікації підпису.
4. Метод **receive_message** дозволяє надіслати серверу відкрито повідомлення **M** та отримати його у зашифрованому вигляді.

5. Метод **receive_message_sign** дозволяє надіслати серверу відкрито повідомлення **M** та отримати відкрито підпис для цього повідомлення.
6. Метод **receive_secret_key** дозволяє надіслати серверу запит на отримання 64 бітного секретного ключа, зашифрованого та підписаного за допомогою параметрів взаємодії.
7. Метод **send_secret_key** дозволяє надіслати серверу секретний ключ, зашифрований та підписаний за допомогою параметрів взаємодії.

В кожному методі результати взаємодії перевіряються та результат перевірки надається користувачу.

In [ ]:
```
class User:
    def __init__(self, p, q, e = 2**16 + 1, serv = Server()):
        print("Initializing user...")
        if not check_prime(p) or not check_prime(q):
            raise RuntimeError("p or q is not a prime number.")
        if math.gcd(e, (p-1)*(q-1)) != 1:
            raise RuntimeError("e is not invertible modulo phi(n)")

        self.serv = serv
        self.p = p
        self.q = q
        self.e = e
        self.n = p*q
        self.d = pow(e, -1, (self.p - 1)*(self.q - 1))
        self.get_server_public_key(KEY_LENGTH * 2)

        print(f"User private key (d, p, q): {(self.d, self.p, self.q)}")
        print(f"User public key (n, e): {(self.n, self.e)}")
        print(f"User server public key (n, e): {(self.serv.n, self.serv.e)}")
        print("------------------------------------------------------------")


    def get_server_public_key(self, len: int):
        self.serv.set_server_key(len)


    def send_message(self, M: str):
        print("Sending message to the server...")
        if int(str2hex(M), 16) > self.serv.n:
            raise RuntimeError("Cannot send the message. Its' length is larger than server's modulo.")

        C = format(pow(int(str2hex(M), 16), self.serv.e, self.serv.n), 'X')
        M1 = self.serv.decrypt(C)

        check = (M1 == M)

        print(f"Sent message: {M}")
        print(f"Sent cyphertext: {C}")
        print(f"Server responce: {M1}")

        if check:
            print("Success")
        else:
            print("Error")
        print("------------------------------------------------------------")


    def send_message_sign(self, M: str):
        print("Sending signature to the server...")

        S = format(pow(int(str2hex(M), 16), self.d, self.n), 'X')
```

```
    check = self.serv.verify(M, S, self.n, self.e)

    print(f"Sent message: {M}")
    print(f"Sent signature: {S}")

    if check:
        print("Success")
    else:
        print("Error")
    print("------------------------------------------------------------")


def receive_message(self, M: str):
    print("Sending request for message to the server...")
    if self.n < int(str2hex(M), 16):
        raise RuntimeError("Cannot receive the message. User's modulo is smaller than message length")

    C = self.serv.encrypt(M, self.n, self.e)
    M1 = num2str(pow(int(C, 16), self.d, self.n))

    check = (M1 == M)

    print(f"Sent message: {M}")
    print(f"Received cyphertext: {C}")
    print(f"Decoded cyphertext: {M1}")

    if check:
        print("Success")
    else:
        print("Error")
    print("------------------------------------------------------------")


def receive_message_sign(self, M: str):
    print("Sending request for message signature to the server...")

    S = self.serv.sign(M)
    M1 = num2str(pow(int(S, 16), self.serv.e, self.serv.n))

    check = (M1 == M)

    print(f"Sent message: {M}")
    print(f"Received signature: {S}")
    print(f"Message signed with signature: {M1}")

    if check:
        print("Success")
    else:
        print("Error")
    print("------------------------------------------------------------")


def receive_secret_key(self):
    print("Sending request for signed secter key to the server...")
    if self.n < self.serv.n:
        raise RuntimeError("Cannot receive the key. User's modulo is smaller than server's.")
```

```
        s_K, s_S = self.serv.sendKey(self.n, self.e)
        K = pow(int(s_K, 16), self.d, self.n)
        S = pow(int(s_S, 16), self.d, self.n)

        check = (pow(S, self.serv.e, self.serv.n) == K)

        print(f"Received key (decoded): {format(K, 'X')}")
        print(f"Received signature (decoded): {format(S, 'X')}")

        print("Verification:")
        if check:
            print("\tSuccess")
        else:
            print("\tError")
        print("-----------------------------------------------------------")



    def send_secret_key(self, K: str):
        print("Sending signed secter key to the server...")
        if self.n > self.serv.n:
            raise RuntimeError("Cannot send key. User's modulo is larger than server's.")
        if int(K, 16) > self.serv.n:
            raise RuntimeError("Cannot send the key. Key length is larger than server's modulo.")

        EK = pow(int(K, 16), self.serv.e, self.serv.n)
        ES = pow(pow(int(K, 16), self.d, self.n), self.serv.e, self.serv.n)

        check = self.serv.receiveKey(format(EK, "X"), format(ES, "X"), self.n, self.e)

        print(f"Sent key: {K}")
        print(f"Sent key (encoded): {format(EK, 'X')}")
        print(f"Sent signature (encoded): {format(ES, 'X')}")

        print("Verification:")
        if check:
            print("\tSuccess")
        else:
            print("\tError")
        print("-----------------------------------------------------------")
```

Симуляція взаємодії

Створення параметрів для приватних ключів

За допомогою генератора сильнопростих чисел згенеруємо два коротких числа (для користувача 1) і два довгих числа (для користувача 2).

In [ ]:
```
print(f"Довжина модуля сервера: {2*KEY_LENGTH}")
print(f"Ключі для користувача 1 ({KEY_LENGTH - 64} бітів для кожного простого):")
p1 = generate_safe_prime(KEY_LENGTH - 64)
print(f"p1 = {p1}")
q1 = generate_safe_prime(KEY_LENGTH - 64, [p1])
print(f"q1 = {q1}")

print()
```

```
print(f"Ключі для користувача 2 ({KEY_LENGTH + 64} бітів для кожного простого):")
p2 = generate_safe_prime(KEY_LENGTH + 64)
print(f"p2 = {p2}")
q2 = generate_safe_prime(KEY_LENGTH + 64, [p2])
print(f"q2 = {q2}")
```

```
Довжина модуля сервера: 512
Ключі для користувача 1 (192 бітів для кожного простого):
p1 = 325194653226342420288253738852229963270764031383154480252 3
q1 = 536790607400852489588660294777721849141662758715851144775 9

Ключі для користувача 2 (320 бітів для кожного простого):
p2 = 139937858734315877978053831242510809457074588538430685454574946772151115664062594608299284842001 9
q2 = 199188218114466056114859978685294726587183540210368528091792673798580536245960223462923698886832 7
```

Взаємодія користувачів

Так як для правильної взаємодії, що включає надсилання зашифрованого повідомлення і підпису для цього повідомлення необхідно, щоб виконувалась умова "ключ надсилача менший за ключ отримувача", ми створимо два користувачі для перевірки правильності взаємодії:

1. Надсилач секретного ключа.
2. Отримувач секретного ключа.

Взаємодія користувача 1 (Надсилач секретного ключа)

In [ ]:
```
u_s = User(p1, q1)

u_s.send_message("Hello_Server1_from_U1")
u_s.send_message_sign("Hello_Server1_from_U1")
u_s.receive_message("Hello_User1_from_Server1")
u_s.receive_message_sign("Hello_User1_from_Server1")
u_s.send_secret_key("dfb0cd9586cf2d9ffff".capitalize())
```

```
Initializing user...
H: Sent request: http://asymcryptwebservice.appspot.com/rsa/serverKey?keySize=512
S: Response: {'modulus': 'CB248FB88EEF90AD3416C2876F1A1C0C07F523FF3163E55875EFB80C2A476C8D0634FC1607DBAC9E2D409049ED69DE367FDBBEE139AF5A42B2E4DDA5DC7DC869', 'publicExponent': '10001'}
User private key (d, p, q): (161520667803570308566796933827109023902954011939511325103828616553019678721513428125150235530568825356049229402649 41, 325194653226342420288253738852229963270764031383154480252 3, 536790607400852489588660294777721849141662758715851144775 9)
User public key (n, e): (174561435428877942523081259415696378679901338706151839734418020940522419765914317597488603036766202810923924858959 57, 65537)
User server public key (n, e): (106394526804301047086035414369260353597079435349211051115800600874766773907603320326293624151777250011209659061729035762146217790512467146382694324275631 13, 65537)
------------------------------------------------------------
Sending message to the server...
H: Sent request: http://asymcryptwebservice.appspot.com/rsa/decrypt?cipherText=3DE30FCCE4568FB55AB710ADE34BEE78517A81D27775C7F9043AF3E60902B053FCE6B67F3F87BEB71FF0E86BBD95A57A9A652E3982AA3 2A7BF8C6F7662FB42B&expectedType=TEXT
S: Response: {'message': 'Hello_Server1_from_U1'}
Sent message: Hello_Server1_from_U1
Sent cyphertext: 3DE30FCCE4568FB55AB710ADE34BEE78517A81D27775C7F9043AF3E60902B053FCE6B67F3F87BEB71FF0E86BBD95A57A9A652E3982AA32A7BF8C6F7662FB42B
Server responce: Hello_Server1_from_U1
Success
------------------------------------------------------------
Sending signature to the server...
H: Sent request: http://asymcryptwebservice.appspot.com/rsa/verify?message=Hello_Server1_from_U1&type=TEXT&signature=61B03E7C3023CEA3AEB697A6CC34FD51C9832DE1A6B319D9B4EDD2BDBA2677BD0B757DA EEF6D72BE439CDCD32D670CDA&modulus=716A33D113790EAA85D222FD40B32E10CBFC9ED54EA12DA34D4678BEA56B12D42A5F2D257FF45FB4B244C809DEDFBF15&publicExponent=10001
S: Response: {'verified': True}
Sent message: Hello_Server1_from_U1
Sent signature: 61B03E7C3023CEA3AEB697A6CC34FD51C9832DE1A6B319D9B4EDD2BDBA2677BD0B757DAEEF6D72BE439CDCD32D670CDA
Success
------------------------------------------------------------
Sending request for message to the server...
H: Sent request: http://asymcryptwebservice.appspot.com/rsa/encrypt?modulus=716A33D113790EAA85D222FD40B32E10CBFC9ED54EA12DA34D4678BEA56B12D42A5F2D257FF45FB4B244C809DEDFBF15&publicExponent= 10001&message=Hello_User1_from_Server1&type=TEXT
S: Response: {'cipherText': '4CBF93642C77BDD014730BD90E6E52ED190D542FC9E4C98578772D9D00C251806D770683D9D30A0273D952CEB178F76B'}
Sent message: Hello_User1_from_Server1
Received cyphertext: 4CBF93642C77BDD014730BD90E6E52ED190D542FC9E4C98578772D9D00C251806D770683D9D30A0273D952CEB178F76B
Decoded cyphertext: Hello_User1_from_Server1
Success
------------------------------------------------------------
Sending request for message signature to the server...
H: Sent request: http://asymcryptwebservice.appspot.com/rsa/sign?message=Hello_User1_from_Server1&type=TEXT
S: Response: {'signature': 'BE9A0564C2F512E1EF443F3142FA3AB0FB7A1CEC5316F16550465444514DAA4CD848788476D2EA12342DA81513C082232EAE8E635407DCB75EE60B061FF57325'}
Sent message: Hello_User1_from_Server1
Received signature: BE9A0564C2F512E1EF443F3142FA3AB0FB7A1CEC5316F16550465444514DAA4CD848788476D2EA12342DA81513C082232EAE8E635407DCB75EE60B061FF57325
Message signed with signature: Hello_User1_from_Server1
Success
------------------------------------------------------------
Sending signed secter key to the server...
H: Sent request: http://asymcryptwebservice.appspot.com/rsa/receiveKey?key=123F5BD82C55FF8617CA909C0EB3EC136E978430122F8E138ED18C3B06A82A39646EE20244190AB86F959ED817D65A03957D2339D0EDD2F87 BE5B668D952A259&signature=31E4548EFB21EDBB65CE33877DAFB929BE5E1EFB970FF6B7AF4AF14B81296AB820F07709448A557F9E40ED2D1D0E5759ED547B2038E0017324F1F29526569B0F&modulus=716A33D113790EAA85D222FD4 0B32E10CBFC9ED54EA12DA34D4678BEA56B12D42A5F2D257FF45FB4B244C809DEDFBF15&publicExponent=10001
S: Response: {'key': '0DFB0CD9586CF2D9FFFF', 'verified': True}
Sent key: Dfb0cd9586cf2d9ffff
Sent key (encoded): 123F5BD82C55FF8617CA909C0EB3EC136E978430122F8E138ED18C3B06A82A39646EE20244190AB86F959ED817D65A03957D2339D0EDD2F87BE5B668D952A259
Sent signature (encoded): 31E4548EFB21EDBB65CE33877DAFB929BE5E1EFB970FF6B7AF4AF14B81296AB820F07709448A557F9E40ED2D1D0E5759ED547B2038E0017324F1F29526569B0F
Verification:
        Success
------------------------------------------------------------
```

Взаємодія користувача 2 (Отримувач секретного ключа)

In [ ]:
```
u_r = User(p2, q2)

u_r.send_message("Hello_Server2_from_U2")
u_s.send_message_sign("Hello_Server2_from_U2")
u_r.receive_message("Hello_User2_from_Server2")
u_r.receive_message_sign("Hello_User2_from_Server2")
u_r.receive_secret_key()
```

```
Initializing user...
H: Sent request: http://asymcryptwebservice.appspot.com/rsa/serverKey?keySize=512
S: Response: {'modulus': 'CB248FB88EEF90AD3416C2876F1A1C0C07F523FF3163E55875EFB80C2A476C8D0634FC1607DBAC9E2D409049ED69DE367FDBBEE139AF5A42B2E4DDA5DC7DC869', 'publicExponent': '10001'}
User private key (d, p, q): (142357712617346254616589497114890463635083215506216885634678728838295837812333382305163205527243558441413598192135738504112206835592680000996942900908345252643160518240465362102138678688797211 7, 139937858734315877978053831242510809457074588538430685454574946772151115664062594608299284842001 9, 199188218114466056114859978685294726587183540210368528091792673798580536245960223462923698886832 7)
User public key (n, e): (278739727280422499740295356350828368296508879167964388092376679868405315727253230442277822611782511441661175055962353029850730272873066433305368765657041417737830774193972657220624555278183821 3, 65537)
User server public key (n, e): (106394526804301047086035414369260353597079435349211051115800600874766773907603320326293624151777250011209659061729035762146217790512467146382694324275631 13, 65537)
------------------------------------------------------------
Sending message to the server...
H: Sent request: http://asymcryptwebservice.appspot.com/rsa/decrypt?cipherText=C7F734503A2F7803AAEAF39ACC501E371C452676A2650596F897506A71A8BCD16ECD9F50EC7E7C1BE59690F8BF2871A8A418FD223203EE753EB2E9B8E2187B51&expectedType=TEXT
S: Response: {'message': 'Hello_Server2_from_U2'}
Sent message: Hello_Server2_from_U2
Sent cyphertext: C7F734503A2F7803AAEAF39ACC501E371C452676A2650596F897506A71A8BCD16ECD9F50EC7E7C1BE59690F8BF2871A8A418FD223203EE753EB2E9B8E2187B51
Server responce: Hello_Server2_from_U2
Success
------------------------------------------------------------
Sending signature to the server...
H: Sent request: http://asymcryptwebservice.appspot.com/rsa/verify?message=Hello_Server2_from_U2&type=TEXT&signature=256A27C3043C54DC418F4370D68D155915C198B8EC784A2952682231D21D915FA8D6B96E942170453646453B82A6B924&modulus=716A33D113790EAA85D222FD40B32E10CBFC9ED54EA12DA34D4678BEA56B12D42A5F2D257FF45FB4B244C809DEDFBF15&publicExponent=10001
S: Response: {'verified': True}
Sent message: Hello_Server2_from_U2
Sent signature: 256A27C3043C54DC418F4370D68D155915C198B8EC784A2952682231D21D915FA8D6B96E942170453646453B82A6B924
Success
------------------------------------------------------------
Sending request for message to the server...
H: Sent request: http://asymcryptwebservice.appspot.com/rsa/encrypt?modulus=9C66D928CE7353E0F38CF4F8CC8DEFCBD6220159B0FE0E5FFB04EC0A0E366BDCF3367170C84338B359E17742C9B3A7222488955C57F26A7FFBF3C04F11EE33DA2C2BA3417A0F6343DF6FE173FE40C785&publicExponent=10001&message=Hello_User2_from_Server2&type=TEXT
S: Response: {'cipherText': '04B84323B4815AC6FBC34925AB933B462A4A265F526103A686578ECF66567F1EDFB3415A2FC66DA523347FFCF32541D9EE74F1979E9A476E32548F1F833F2B2E856180595F79519CD6D558AFF6EE15BC'}
Sent message: Hello_User2_from_Server2
Received cyphertext: 04B84323B4815AC6FBC34925AB933B462A4A265F526103A686578ECF66567F1EDFB3415A2FC66DA523347FFCF32541D9EE74F1979E9A476E32548F1F833F2B2E856180595F79519CD6D558AFF6EE15BC
Decoded cyphertext: Hello_User2_from_Server2
Success
------------------------------------------------------------
Sending request for message signature to the server...
H: Sent request: http://asymcryptwebservice.appspot.com/rsa/sign?message=Hello_User2_from_Server2&type=TEXT
S: Response: {'signature': 'ABD785CAD75A2721A2EFBF8954F18ADBD9F5F6DBECD9C52B69964F7DFA1107C1812A48F903F8C7E6A8EE5EADE392CF23A9A739D8C506BF33F892BEE894C380A9'}
Sent message: Hello_User2_from_Server2
Received signature: ABD785CAD75A2721A2EFBF8954F18ADBD9F5F6DBECD9C52B69964F7DFA1107C1812A48F903F8C7E6A8EE5EADE392CF23A9A739D8C506BF33F892BEE894C380A9
Message signed with signature: Hello_User2_from_Server2
Success
------------------------------------------------------------
Sending request for signed secter key to the server...
Sent request: http://asymcryptwebservice.appspot.com/rsa/sendKey?modulus=9C66D928CE7353E0F38CF4F8CC8DEFCBD6220159B0FE0E5FFB04EC0A0E366BDCF3367170C84338B359E17742C9B3A7222488955C57F26A7FFBF3C04F11EE33DA2C2BA3417A0F6343DF6FE173FE40C785&publicExponent=10001
response: {'key': '24C9C9683D3194B08CC8E18BB917494A0E31F56B9E1979D75A21FE36CCCE6451E078FA17DA0146C5816662540DEC3DB446790BAA1D727288F9BF8BFC4751002FC0CE5342FEE0A4751051D4ABB2A3BDCF', 'signature': '1A4668F96D5795C9DB48C991BBE9FCAB02BFFBF7ADD8B4D2FAA14F046BB025497A569FEF05C90B83AF0B1F9F67C2F978532296F28353ACB7F6200338FBA85F6E7EF50471B90AF0024AEE928E12C6FCE0'}
Received key (decoded): 8F0F83106C7711ED
Received signature (decoded): 92088855FECD1AAF044CF08857DE7276C7F39DDC4D226ECDB693FCEFBC8B3EB533FE8C2AA814E7C37F35620C8A703B3969024DC7F6F37F0A16CFB71132942928
Verification:
        Success
------------------------------------------------------------
```