

## Задача

Необходимо создать приложение типа клиент-сервер, которое будет осуществлять передачу сообщений между собой с помощью различных алгоритмов шифрования.

## Решение

Два приложения, написанные на языке Python, запущены в двух терминалах и совершают обмен сообщениями.

Для установки соединения один терминал выступает в качестве сервера и открывает порт на прием подключений, в то время как другой терминал подключается по известному порту. Два основных кода соответственно для сервера и клиента для настройки соединения, а также обмена сообщениями представлены ниже.

Листинг 1 – Сервер, подключение клиентов и обмен сообщениями.

```
def run_server(port=4321):
    serv_sock = create_serv_sock(port)
    cid = 0
    while True:
        client_sock = accept_client_conn(serv_sock, cid)
        serve_client(client_sock, cid)
        cid += 1

def create_serv_sock(serv_port):
    serv_sock = socket.socket(socket.AF_INET,
                              socket.SOCK_STREAM,
                              proto=0)
    serv_sock.bind(('', serv_port))
    serv_sock.listen()
    return serv_sock

def accept_client_conn(serv_sock, cid):
    client_sock, client_addr = serv_sock.accept()
    print(f'Client #{cid} connected '
          f'{client_addr[0]}:{client_addr[1]}')
    return client_sock

def serve_client(client_sock, cid):
    key = DH(client_sock)
    rsa_keys = rsa_init(client_sock, key).split(':')
    e = int(rsa_keys[0])
    d = int(rsa_keys[1])
    n = int(rsa_keys[2])
    print(f'Connection established')

    while True:
        request = client_sock.recv(1024).decode()
        if not request:
            print(f'Client #{cid} disconnected')
```

```

        client_sock.close()
        break
    else:
        text = rsa_decrypt(request, d, n)
        print('Received', text)
        message = input()
        client_sock.sendall(rsa_encrypt(message, e, n))
        print(f'Message was sent')

if __name__ == '__main__':
    run_server(port=int(sys.argv[1]))

```

## Листинг 2 – Клиент, подключение и обмен сообщениями.

```

client_sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
client_sock.connect(('127.0.0.1', 4321))
key = DH(client_sock)
rsa_keys = rsa_init(client_sock, key).split(':')
e = int(rsa_keys[0])
d = int(rsa_keys[1])
n = int(rsa_keys[2])
print(f'Enter your message')

while True:
    message = input()
    if message == 'stop':
        break
    client_sock.sendall(rsa_encrypt(message, e, n))
    print(f'Message was sent')
    data = client_sock.recv(1024).decode()
    print('Received', rsa_decrypt(data, d, n))
client_sock.close()

```

Как видно из кода, после подключения первое сообщение отправляет клиент, после чего переходит в режим прослушивания. В свою очередь сервер сперва ждет сообщение, после чего может отправить сообщение клиенту и снова переходит в режим прослушивания. Таким образом участники общаются поочередно.

Для обеспечения безопасности общения используется 3 алгоритма. Для начала реализуется обмен ключами с помощью алгоритма Диффи-Хеллмана. Для него клиент вручную вводит простое открытое число  $p$ .

### Листинг 3 – Алгоритм Диффи-Хеллмана на стороне клиента.

```
def gcd(a,b):
    while a != b:
        if a > b:
            a = a - b
        else:
            b = b - a
    return a

def primitive_root(modulo):
    required_set = set(num for num in range (1, modulo) if gcd(num, modulo)
== 1)
    for g in range(1, modulo):
        actual_set = set(pow(g, powers) % modulo for powers in range (1,
modulo))
        if required_set == actual_set:
            return g

def Generate():
    return randint(1000, 10000)

def IsPrime(num):
    for n in range(2,int(num**1/2)+1):
        if num%n==0:
            return False
    return True

def DH(client_sock):
    while True:
        print(f'Enter primary number p for DH')
        p = int(input())
        if IsPrime(p):
            break
        else:
            print(f'p is not primary')
    g = primitive_root(p)
    secret = Generate()
    self_open = (g ** secret) % p
    deliver = str(p) + ":" + str(g) + ":" + str(self_open)
    client_sock.sendall(deliver.encode())
    another_open = client_sock.recv(1024).decode()
    key = (int(another_open) ** secret) % p
    return str(key).encode()
```

### Листинг 4 – Алгоритм Диффи-Хеллмана на стороне сервера.

```
def Generate():
    return randint(1000, 10000)

def IsPrime(num):
    for n in range(2,int(num**1/2)+1):
```

```

        if num%n==0:
            return False
    return True

def DH(client_sock):
    secret = Generate()
    deliver = (client_sock.recv(1024).decode()).split(':')
    self_open = (int(deliver[1]) ** secret) % int(deliver[0])
    client_sock.sendall(str(self_open).encode())
    key = (int(deliver[2]) ** secret) % int(deliver[0])
    return str(key).encode()

```

После того, как у обеих сторон есть общий ключ, мы можем использовать его для симметричного шифрования AES.

#### Листинг 5 – Алгоритм AES.

```

s_box = (
    0x63, 0x7C, 0x77, 0x7B, 0xF2, 0x6B, 0x6F, 0xC5, 0x30, 0x01, 0x67, 0x2B,
    0xFE, 0xD7, 0xAB, 0x76,
    0xCA, 0x82, 0xC9, 0x7D, 0xFA, 0x59, 0x47, 0xF0, 0xAD, 0xD4, 0xA2, 0xAF,
    0x9C, 0xA4, 0x72, 0xC0,
    0xB7, 0xFD, 0x93, 0x26, 0x36, 0x3F, 0xF7, 0xCC, 0x34, 0xA5, 0xE5, 0xF1,
    0x71, 0xD8, 0x31, 0x15,
    0x04, 0xC7, 0x23, 0xC3, 0x18, 0x96, 0x05, 0x9A, 0x07, 0x12, 0x80, 0xE2,
    0xEB, 0x27, 0xB2, 0x75,
    0x09, 0x83, 0x2C, 0x1A, 0x1B, 0x6E, 0x5A, 0xA0, 0x52, 0x3B, 0xD6, 0xB3,
    0x29, 0xE3, 0x2F, 0x84,
    0x53, 0xD1, 0x00, 0xED, 0x20, 0xFC, 0xB1, 0x5B, 0x6A, 0xCB, 0xBE, 0x39,
    0x4A, 0x4C, 0x58, 0xCF,
    0xD0, 0xEF, 0xAA, 0xFB, 0x43, 0x4D, 0x33, 0x85, 0x45, 0xF9, 0x02, 0x7F,
    0x50, 0x3C, 0x9F, 0xA8,
    0x51, 0xA3, 0x40, 0x8F, 0x92, 0x9D, 0x38, 0xF5, 0xBC, 0xB6, 0xDA, 0x21,
    0x10, 0xFF, 0xF3, 0xD2,
    0xCD, 0x0C, 0x13, 0xEC, 0x5F, 0x97, 0x44, 0x17, 0xC4, 0xA7, 0x7E, 0x3D,
    0x64, 0x5D, 0x19, 0x73,
    0x60, 0x81, 0x4F, 0xDC, 0x22, 0x2A, 0x90, 0x88, 0x46, 0xEE, 0xB8, 0x14,
    0xDE, 0x5E, 0x0B, 0xDB,
    0xE0, 0x32, 0x3A, 0x0A, 0x49, 0x06, 0x24, 0x5C, 0xC2, 0xD3, 0xAC, 0x62,
    0x91, 0x95, 0xE4, 0x79,
    0xE7, 0xC8, 0x37, 0x6D, 0x8D, 0xD5, 0x4E, 0xA9, 0x6C, 0x56, 0xF4, 0xEA,
    0x65, 0x7A, 0xAE, 0x08,
    0xBA, 0x78, 0x25, 0x2E, 0x1C, 0xA6, 0xB4, 0xC6, 0xE8, 0xDD, 0x74, 0x1F,
    0x4B, 0xBD, 0x8B, 0x8A,
    0x70, 0x3E, 0xB5, 0x66, 0x48, 0x03, 0xF6, 0x0E, 0x61, 0x35, 0x57, 0xB9,
    0x86, 0xC1, 0x1D, 0x9E,
    0xE1, 0xF8, 0x98, 0x11, 0x69, 0xD9, 0x8E, 0x94, 0x9B, 0x1E, 0x87, 0xE9,
    0xCE, 0x55, 0x28, 0xDF,
    0x8C, 0xA1, 0x89, 0x0D, 0xBF, 0xE6, 0x42, 0x68, 0x41, 0x99, 0x2D, 0x0F,
    0xB0, 0x54, 0xBB, 0x16,
)

```

```

inv_s_box = (
    0x52, 0x09, 0x6A, 0xD5, 0x30, 0x36, 0xA5, 0x38, 0xBF, 0x40, 0xA3, 0x9E,
    0x81, 0xF3, 0xD7, 0xFB,
    0x7C, 0xE3, 0x39, 0x82, 0x9B, 0x2F, 0xFF, 0x87, 0x34, 0x8E, 0x43, 0x44,
    0xC4, 0xDE, 0xE9, 0xCB,
    0x54, 0x7B, 0x94, 0x32, 0xA6, 0xC2, 0x23, 0x3D, 0xEE, 0x4C, 0x95, 0x0B,
    0x42, 0xFA, 0xC3, 0x4E,
    0x08, 0x2E, 0xA1, 0x66, 0x28, 0xD9, 0x24, 0xB2, 0x76, 0x5B, 0xA2, 0x49,
    0x6D, 0x8B, 0xD1, 0x25,
    0x72, 0xF8, 0xF6, 0x64, 0x86, 0x68, 0x98, 0x16, 0xD4, 0xA4, 0x5C, 0xCC,
    0x5D, 0x65, 0xB6, 0x92,
    0x6C, 0x70, 0x48, 0x50, 0xFD, 0xED, 0xB9, 0xDA, 0x5E, 0x15, 0x46, 0x57,
    0xA7, 0x8D, 0x9D, 0x84,
    0x90, 0xD8, 0xAB, 0x00, 0x8C, 0xBC, 0xD3, 0x0A, 0xF7, 0xE4, 0x58, 0x05,
    0xB8, 0xB3, 0x45, 0x06,
    0xD0, 0x2C, 0x1E, 0x8F, 0xCA, 0x3F, 0x0F, 0x02, 0xC1, 0xAF, 0xBD, 0x03,
    0x01, 0x13, 0x8A, 0x6B,
    0x3A, 0x91, 0x11, 0x41, 0x4F, 0x67, 0xDC, 0xEA, 0x97, 0xF2, 0xCF, 0xCE,
    0xF0, 0xB4, 0xE6, 0x73,
    0x96, 0xAC, 0x74, 0x22, 0xE7, 0xAD, 0x35, 0x85, 0xE2, 0xF9, 0x37, 0xE8,
    0x1C, 0x75, 0xDF, 0x6E,
    0x47, 0xF1, 0x1A, 0x71, 0x1D, 0x29, 0xC5, 0x89, 0x6F, 0xB7, 0x62, 0x0E,
    0xAA, 0x18, 0xBE, 0x1B,
    0xFC, 0x56, 0x3E, 0x4B, 0xC6, 0xD2, 0x79, 0x20, 0x9A, 0xDB, 0xC0, 0xFE,
    0x78, 0xCD, 0x5A, 0xF4,
    0x1F, 0xDD, 0xA8, 0x33, 0x88, 0x07, 0xC7, 0x31, 0xB1, 0x12, 0x10, 0x59,
    0x27, 0x80, 0xEC, 0x5F,
    0x60, 0x51, 0x7F, 0xA9, 0x19, 0xB5, 0x4A, 0x0D, 0x2D, 0xE5, 0x7A, 0x9F,
    0x93, 0xC9, 0x9C, 0xEF,
    0xA0, 0xE0, 0x3B, 0x4D, 0xAE, 0x2A, 0xF5, 0xB0, 0xC8, 0xEB, 0xBB, 0x3C,
    0x83, 0x53, 0x99, 0x61,
    0x17, 0x2B, 0x04, 0x7E, 0xBA, 0x77, 0xD6, 0x26, 0xE1, 0x69, 0x14, 0x63,
    0x55, 0x21, 0x0C, 0x7D,
)

```

```

def sub_bytes(s):
    for i in range(4):
        for j in range(4):
            s[i][j] = s_box[s[i][j]]

```

```

def inv_sub_bytes(s):
    for i in range(4):
        for j in range(4):
            s[i][j] = inv_s_box[s[i][j]]

```

```

def shift_rows(s):
    s[0][1], s[1][1], s[2][1], s[3][1] = s[1][1], s[2][1], s[3][1], s[0][1]
    s[0][2], s[1][2], s[2][2], s[3][2] = s[2][2], s[3][2], s[0][2], s[1][2]
    s[0][3], s[1][3], s[2][3], s[3][3] = s[3][3], s[0][3], s[1][3], s[2][3]

```

```

def inv_shift_rows(s):
    s[0][1], s[1][1], s[2][1], s[3][1] = s[3][1], s[0][1], s[1][1], s[2][1]

```

```

s[0][2], s[1][2], s[2][2], s[3][2] = s[2][2], s[3][2], s[0][2], s[1][2]
s[0][3], s[1][3], s[2][3], s[3][3] = s[1][3], s[2][3], s[3][3], s[0][3]

def add_round_key(s, k):
    for i in range(4):
        for j in range(4):
            s[i][j] ^= k[i][j]

xtime = lambda a: (((a << 1) ^ 0x1B) & 0xFF) if (a & 0x80) else (a << 1)

def mix_single_column(a):
    t = a[0] ^ a[1] ^ a[2] ^ a[3]
    u = a[0]
    a[0] ^= t ^ xtime(a[0] ^ a[1])
    a[1] ^= t ^ xtime(a[1] ^ a[2])
    a[2] ^= t ^ xtime(a[2] ^ a[3])
    a[3] ^= t ^ xtime(a[3] ^ u)

def mix_columns(s):
    for i in range(4):
        mix_single_column(s[i])

def inv_mix_columns(s):
    for i in range(4):
        u = xtime(xtime(s[i][0] ^ s[i][2]))
        v = xtime(xtime(s[i][1] ^ s[i][3]))
        s[i][0] ^= u
        s[i][1] ^= v
        s[i][2] ^= u
        s[i][3] ^= v

    mix_columns(s)

r_con = (
    0x00, 0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40,
    0x80, 0x1B, 0x36, 0x6C, 0xD8, 0xAB, 0x4D, 0x9A,
    0x2F, 0x5E, 0xBC, 0x63, 0xC6, 0x97, 0x35, 0x6A,
    0xD4, 0xB3, 0x7D, 0xFA, 0xEF, 0xC5, 0x91, 0x39,
)

def bytes2matrix(text):
    return [list(text[i:i+4]) for i in range(0, len(text), 4)]

def matrix2bytes(matrix):
    return bytes(sum(matrix, []))

def xor_bytes(a, b):
    return bytes(i^j for i, j in zip(a, b))

def inc_bytes(a):

```

```

out = list(a)
for i in reversed(range(len(out))):
    if out[i] == 0xFF:
        out[i] = 0
    else:
        out[i] += 1
        break
return bytes(out)

def pad(plaintext):
    padding_len = 16 - (len(plaintext) % 16)
    padding = bytes([padding_len] * padding_len)
    return plaintext + padding

def unpad(plaintext):
    padding_len = plaintext[-1]
    assert padding_len > 0
    message, padding = plaintext[:-padding_len], plaintext[-padding_len:]
    assert all(p == padding_len for p in padding)
    return message

def split_blocks(message, block_size=16, require_padding=True):
    assert len(message) % block_size == 0 or not require_padding
    return [message[i:i+16] for i in range(0, len(message), block_size)]

class AES:
    rounds_by_key_size = {16: 10, 24: 12, 32: 14}
    def __init__(self, master_key):
        assert len(master_key) in AES.rounds_by_key_size
        self.n_rounds = AES.rounds_by_key_size[len(master_key)]
        self._key_matrices = self._expand_key(master_key)

    def _expand_key(self, master_key):
        key_columns = bytes2matrix(master_key)
        iteration_size = len(master_key) // 4

        i = 1
        while len(key_columns) < (self.n_rounds + 1) * 4:
            word = list(key_columns[-1])

            if len(key_columns) % iteration_size == 0:
                word.append(word.pop(0))
                word = [s_box[b] for b in word]
                word[0] ^= r_con[i]
                i += 1
            elif len(master_key) == 32 and len(key_columns) % iteration_size
== 4:
                word = [s_box[b] for b in word]

            word = xor_bytes(word, key_columns[-iteration_size])
            key_columns.append(word)

        return [key_columns[4*i : 4*(i+1)] for i in range(len(key_columns) //
4)]

```

```

def encrypt_block(self, plaintext):
    assert len(plaintext) == 16

    plain_state = bytes2matrix(plaintext)

    add_round_key(plain_state, self._key_matrices[0])

    for i in range(1, self.n_rounds):
        sub_bytes(plain_state)
        shift_rows(plain_state)
        mix_columns(plain_state)
        add_round_key(plain_state, self._key_matrices[i])

    sub_bytes(plain_state)
    shift_rows(plain_state)
    add_round_key(plain_state, self._key_matrices[-1])

    return matrix2bytes(plain_state)

def decrypt_block(self, ciphertext):
    assert len(ciphertext) == 16

    cipher_state = bytes2matrix(ciphertext)

    add_round_key(cipher_state, self._key_matrices[-1])
    inv_shift_rows(cipher_state)
    inv_sub_bytes(cipher_state)

    for i in range(self.n_rounds - 1, 0, -1):
        add_round_key(cipher_state, self._key_matrices[i])
        inv_mix_columns(cipher_state)
        inv_shift_rows(cipher_state)
        inv_sub_bytes(cipher_state)

    add_round_key(cipher_state, self._key_matrices[0])

    return matrix2bytes(cipher_state)

def encrypt_cbc(self, plaintext, iv):
    assert len(iv) == 16

    plaintext = pad(plaintext)

    blocks = []
    previous = iv
    for plaintext_block in split_blocks(plaintext):
        block = self.encrypt_block(xor_bytes(plaintext_block, previous))
        blocks.append(block)
        previous = block

    return b''.join(blocks)

def decrypt_cbc(self, ciphertext, iv):
    assert len(iv) == 16

    blocks = []

```



```

        previous = iv
        for ciphertext_block in split_blocks(ciphertext):
            blocks.append(xor_bytes(previous,
self.decrypt_block(ciphertext_block)))
            previous = ciphertext_block

    return unpad(b''.join(blocks))

AES_KEY_SIZE = 16
HMAC_KEY_SIZE = 16
IV_SIZE = 16
SALT_SIZE = 16
HMAC_SIZE = 32

def get_key_iv(password, salt, workload=100000):
    stretched = pbkdf2_hmac('sha256', password, salt, workload, AES_KEY_SIZE
+ IV_SIZE + HMAC_KEY_SIZE)
    aes_key, stretched = stretched[:AES_KEY_SIZE], stretched[AES_KEY_SIZE:]
    hmac_key, stretched = stretched[:HMAC_KEY_SIZE],
stretched[HMAC_KEY_SIZE:]
    iv = stretched[:IV_SIZE]
    return aes_key, hmac_key, iv

def encrypt(key, plaintext, workload=100000):
    if isinstance(key, str):
        key = key.encode('utf-8')
    if isinstance(plaintext, str):
        plaintext = plaintext.encode('utf-8')

    salt = os.urandom(SALT_SIZE)
    key, hmac_key, iv = get_key_iv(key, salt, workload)
    ciphertext = AES(key).encrypt_cbc(plaintext, iv)
    hmac = new_hmac(hmac_key, salt + ciphertext, 'sha256').digest()
    assert len(hmac) == HMAC_SIZE

    return hmac + salt + ciphertext

def decrypt(key, ciphertext, workload=100000):

    assert len(ciphertext) % 16 == 0

    assert len(ciphertext) >= 32

    if isinstance(key, str):
        key = key.encode('utf-8')

    hmac, ciphertext = ciphertext[:HMAC_SIZE], ciphertext[HMAC_SIZE:]
    salt, ciphertext = ciphertext[:SALT_SIZE], ciphertext[SALT_SIZE:]
    key, hmac_key, iv = get_key_iv(key, salt, workload)

    expected_hmac = new_hmac(hmac_key, salt + ciphertext, 'sha256').digest()
    assert compare_digest(hmac, expected_hmac)

    return AES(key).decrypt_cbc(ciphertext, iv)

```

Алгоритм AES используется для шифрования и безопасной передачи ключей алгоритма RSA, для того чтобы оба пользователя могли шифровать и дешифровать сообщения этим алгоритмом.

Алгоритм RSA выглядит одинаково для клиента и сервера, за исключением функции инициализации, которая запускает генерацию ключей на основе введенных клиентом простых чисел  $p$  и  $q$  и передает их в зашифрованном виде серверу.

#### Листинг 6 – Алгоритм RSA.

```
alf = "абвгдежзийклмнопрстуфхцчшщъыьэюя"
let = ["А", "Б", "В", "Г", "Д", "Е", "Ж", "З", "И", "Й", "К", "Л", "М",
"Н", "О", "П", "Р", "С", "Т", "У", "Ф", "Х", "Ц", "Ч", "Ш", "Щ", "Ъ",
"Ы", "Ь", "Э", "Ю", "Я"]

for i in range(len(let)):
    let.append(alf[i])

cod = []
for i in range(192, 256):
    cod.append(i)

def chace(a, b):
    arr = []
    if a < b:
        arr.append(0)
    while a != 0 and b != 0:
        if a > b:
            arr.append(a // b)
            a = a % b
        else:
            arr.append(b // a)
            b = b % a
    return arr

def find(arr, c):
    for i in range(len(arr)):
        if arr[i] == c:
            return i

def add(s):
    while len(s) < 5:
        s = '0' + s
    return s

def delete(s):
    while s[0] == '0':
        s = s[1:len(s)]
    return s
```

```

def rsa_encrypt(S, E, N):
    s = str()
    ciph = str()
    for i in range(len(S)):
        s += str(cod[find(let, S[i])])
        ciph += add(str(pow(cod[find(let, S[i])], E, N)))
    return ciph.encode()

def rsa_decrypt(S, d, N):
    deciph = ''
    for i in range(0, len(S), 5):
        num = int(delete(S[i: i + 5]))
        num = pow(num, d, N)
        deciph += let[find(cod, num)]
    return deciph

```

Листинг 7 – Инициализация RSA на стороне клиента.

```

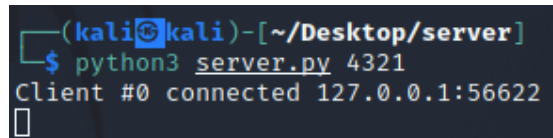
def rsa_init(client_sock, key):
    while True:
        print(f'Enter primary numbers p for RSA')
        p = int(input())
        print(f'Enter primary numbers q for RSA')
        q = int(input())
        if IsPrime(p):
            if IsPrime(q):
                break
            else:
                print(f'q is not primary')
        else:
            print(f'p is not primary')
    N = p * q
    fi = (p - 1) * (q - 1)
    while True:
        E = randint(2, fi)
        if gcd(E, fi) == 1:
            break
    dr = chace(fi, E)
    p = [1, dr[0]]
    for i in range(2, len(dr) + 1):
        p.append(dr[i - 1] * p[i - 1] + p[i - 2])
    d = (1**((len(dr) - 1) * p[len(p) - 2]))
    d = fi - d
    deliver = str(E) + ":" + str(d) + ":" + str(N)
    client_sock.sendall(encrypt(key, deliver.encode()))
    return deliver

```

## Листинг 8 – Инициализация RSA на стороне сервера.

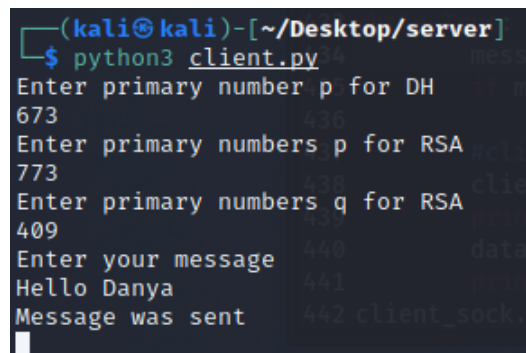
```
def rsa_init(client_sock, key):  
    deliver = client_sock.recv(1024)  
    deliver = (decrypt(key, deliver)).decode()  
    return deliver
```

## Работа программы



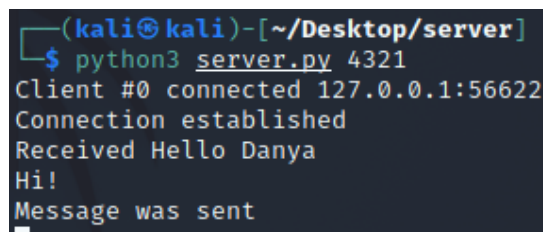
```
(kali㉿kali)-[~/Desktop/server]  
$ python3 server.py 4321  
Client #0 connected 127.0.0.1:56622  
█
```

Рисунок 1 – старт сервера, подключение клиента.



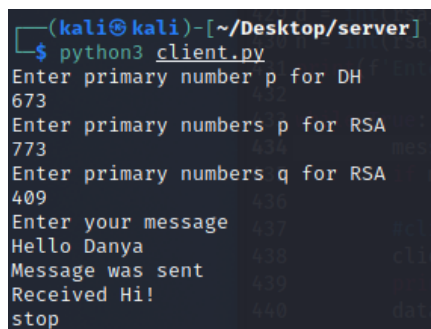
```
(kali㉿kali)-[~/Desktop/server]  
$ python3 client.py  
Enter primary number p for DH  
673  
Enter primary numbers p for RSA  
773  
Enter primary numbers q for RSA  
409  
Enter your message  
Hello Danya  
Message was sent
```

Рисунок 2 – подключение клиента, ввод необходимых чисел для генерации ключей, отправка первого сообщения.



```
(kali㉿kali)-[~/Desktop/server]  
$ python3 server.py 4321  
Client #0 connected 127.0.0.1:56622  
Connection established  
Received Hello Danya  
Hi!  
Message was sent
```

Рисунок 3 – получение первого сообщения на сервере, отправление ответного сообщения.



```
(kali㉿kali)-[~/Desktop/server]  
$ python3 client.py  
Enter primary number p for DH  
673  
Enter primary numbers p for RSA  
773  
Enter primary numbers q for RSA  
409  
Enter your message  
Hello Danya  
Message was sent  
Received Hi!  
stop
```

Рисунок 4 – получение ответного сообщения от сервера, использование команды stop для отключения от сервера.