Клиент и сервер являются двумя различными docker-контейнерами и общаются в одной сети.

Серверная часть программы хранит файлы:

def assert\_system\_file(file: str):

if file.find("..") != -1:

raise PermissionError

if re.match(r'/\*\.', file) is not None:

raise PermissionError

def get\_for\_user(user: str, file: str) -> bytes:

assert\_system\_file(file)

f = open(STORAGE\_PATH + '/' + user + '/' + file, 'rb')

data = f.read()

f.close()

return data

def create\_for\_user(user: str, file: str, data: bytes):

assert\_system\_file(file)

f = open(STORAGE\_PATH + '/' + user + '/' + file, 'wb')

f.write(data)

f.close()

def delete\_for\_user(user: str, file: str):

assert\_system\_file(file)

os.remove(STORAGE\_PATH + '/' + user + '/' + file)

Генерирует случайный сеансовый ключ и выполняет аутентификацию по паролю:

def authenticate(self, is\_first=False):

if is\_first:

self.rsa\_pub = self.sock.recv().body

self.key = os.urandom(16)

encoded\_key = crypt.encrypt\_rsa(self.rsa\_pub, self.key)

self.sock.send(encoded\_key)

self.last\_token\_update = datetime.datetime.now()

if is\_first:

self.sock.send\_string("Do you want to login or signup? Press (l/s)")

authtype = decode\_utf8(self.sock.recv().body)

self.sock.send\_string("Enter login:")

self.username = decode\_utf8(self.sock.recv().body)

print("{} trying to authenticate".format(self.username))

self.sock.send\_string("Enter password:")

password = decrypt\_aes(self.key, self.sock.recv().body)

if is\_first and authtype == 's':

storage.signup(self.username, password)

else:

if not storage.authenticate(self.username, password):

print("Invalid login {} supplied. Closing the socket.".format(

self.username))

self.sock.send(b'', input\_wanted=NO\_INPUT,

response\_code=FORBIDDEN)

self.sock.close()

return

self.sock.send(b'', input\_wanted=NO\_INPUT, response\_code=SUCCESS)

У ключа есть срок годности (1 минута):

def refresh\_token(self):

if datetime.datetime.now() < self.token\_timeout + self.last\_token\_update:

return

self.sock.send(b'', response\_code=REFRESH)

self.authenticate()

Отправляет клиенту зашифрованный ключ:

if is\_first:

self.rsa\_pub = self.sock.recv().body

self.key = os.urandom(16)

encoded\_key = crypt.encrypt\_rsa(self.rsa\_pub, self.key)

self.sock.send(encoded\_key)

Клиент отправляет серверу открытый ключ:

def start(self):

client\_storage.gen\_rsa()

self.sock = Socket(socket.create\_connection((self.address, self.port)))

if self.auth(is\_first=True):

self.receive\_loop()

def gen\_rsa():

pub, private = crypt.generate\_rsa\_keypair()

pub\_f = open(STORAGE\_PATH + '/' + RSA\_PUB\_FILE, 'wb')

private\_f = open(STORAGE\_PATH + '/' + RSA\_PRI\_FILE, 'wb')

pub\_f.write(pub)

private\_f.write(private)

pub\_f.close()

private\_f.close()

Расшифровывает сеансовый ключ:

def auth(self, is\_first=False, is\_refresh=False):

pub, private = client\_storage.get\_rsa\_pair()

if is\_first:

self.sock.send(pub)

aes\_key\_coded = self.sock.recv().body

self.aes\_key = crypt.decrypt\_rsa(private, aes\_key\_coded)

if is\_first:

print(decode\_utf8(self.sock.recv().body))

authtype = input()

self.sock.send\_string(authtype)

print(decode\_utf8(self.sock.recv().body))

username = input()

self.sock.send\_string(username)

print(decode\_utf8(self.sock.recv().body))

password = getpass()

self.sock.send(encrypt\_aes(self.aes\_key, encode\_utf8(password)))

response = self.sock.recv()

if response.response\_code != SUCCESS:

print("Failed to authenticate.")

self.sock.close()

return False

return True

Расшифровывает текст:

def decrypt\_aes(key: bytes, data: bytes) -> bytes:

iv = data[:AES.block\_size]

padded = AES.new(key, iv=iv, mode=AES.MODE\_OFB).decrypt(data[AES.block\_size:])

return unpad(padded, block\_size=AES.block\_size)