
Bezpieczeństwo Systemów Komputerowych

Raport projektu

Autorzy:
Wiktor Krasiński 179987
Przemysław Rośleń 180150

Wersja: 2.0

Wersje

Wersje	Data	Opis zmian
1.0	26.04.2022	Stworzenie połowicznego raportu
2.0	11.06.2022	Stworzenie raportu końcowego

1. Projekt – termin kontrolny

1.1 Wykorzystywane narzędzia

Projekt zaimplementowaliśmy w języku Python. Do stworzenia graficznego interfejsu użytkownika wykorzystaliśmy bibliotekę Tkinter oraz jej rozszerzenie Customtkinter. Do operacji kryptograficznych wykorzystaliśmy bibliotekę Pycryptodome oraz moduły Cryptography i Secrets. Do współdzielenia kodu wykorzystaliśmy platformę Github (adres repozytorium: <https://github.com/przeros/Encrypted-Messaging-App>). Do połączenia sieciowego poprzez gniazda sieciowe wykorzystaliśmy bibliotekę Socket. Żeby aplikacja działała płynnie i interfejs użytkownika działał współbieżnie z komponentami logiki biznesowej, zaprojektowaliśmy architekturę wielowątkową poprzez użycie modułu threading.

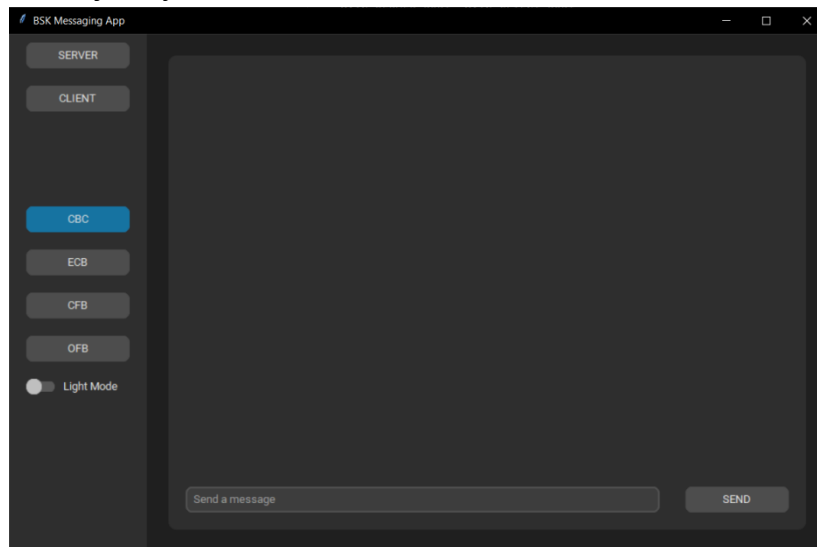
1.2 Funkcjonalności

Udało nam się do tej pory zaimplementować następujące funkcjonalności:

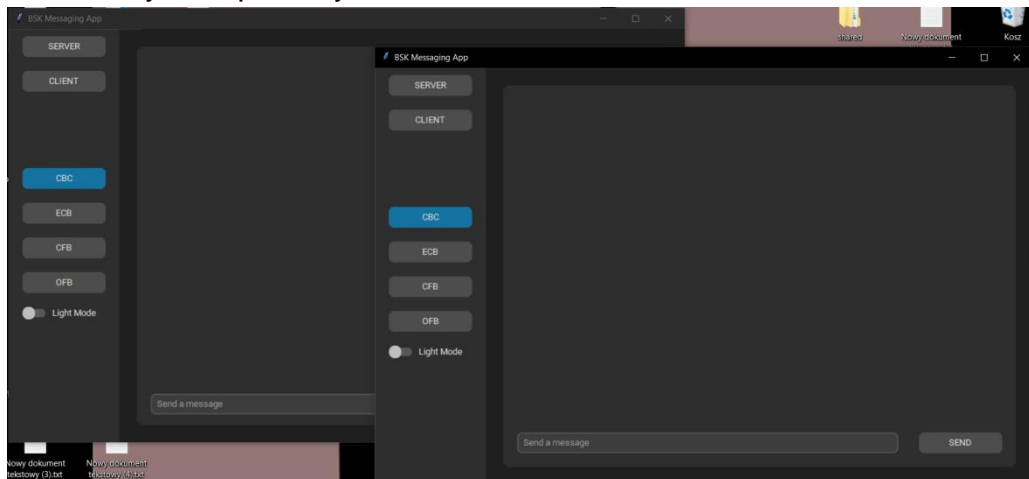
- Interfejs użytkownika
- Dwa identyczne procesy
- Zestawienie połączenia sieciowego poprzez gniazda sieciowe z wykorzystaniem protokołu TCP/IP między procesami
- Tworzenie pary kluczy: prywatnych i publicznych z użyciem RSA
- Zapis pary kluczy do plików, w tym prywatnego w formie zaszyfrowanej używając algorytmu symetrycznego AES w trybie CBC, który jako klucz przyjął hash hasła utworzony poprzez SHA3 w trybie 256 bitowym
- Wymiana kluczy publicznych między procesami
- Generacja klucza sesyjnego
- Zaszyfrowanie klucza sesyjnego przez serwer kluczem publicznym klienta
- Bezpieczne wysłanie klucza sesyjnego klientowi
- Wysyłanie i odbieranie wiadomości oraz potwierdzanie ich odbioru przez serwer
- Szyfrowanie i odszyfrowywanie wiadomości algorytmem symetrycznym AES z kluczem sesyjnym
- Możliwość wyboru trybu szyfrowania algorytmu AES: CBC, ECB, CFB, OFB

1.3 Zdjęcia programu

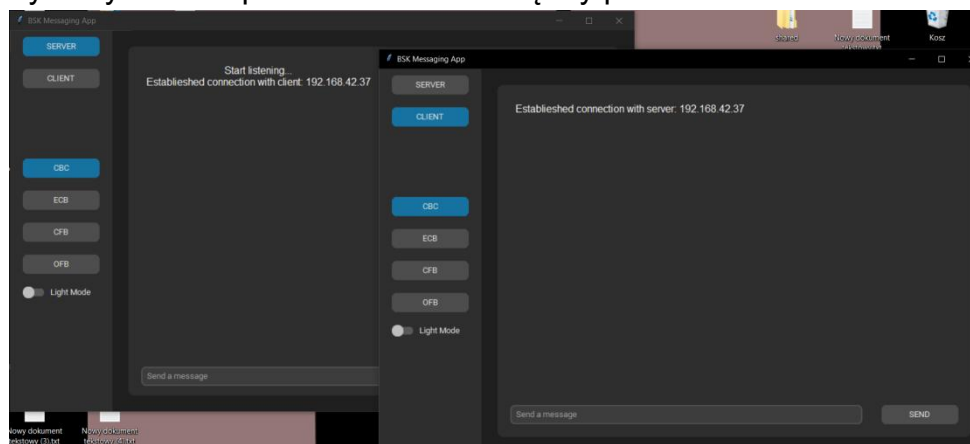
- Interfejs użytkownika



- Dwa identyczne procesy



- Zestawienie połączenia sieciowego poprzez gniazda sieciowe z wykorzystaniem protokołu TCP/IP między procesami

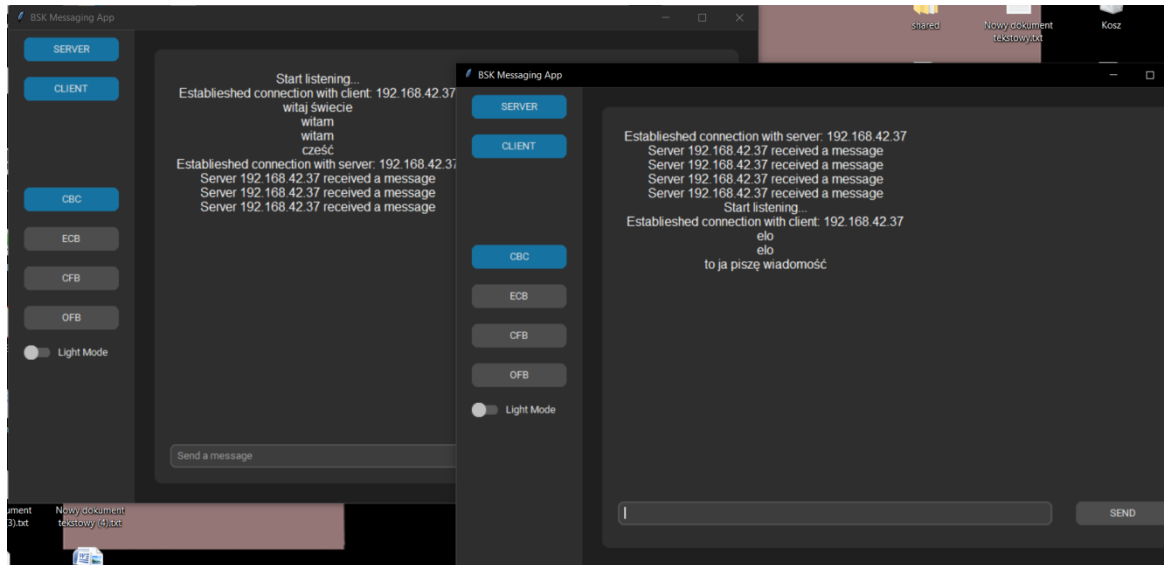


- Tworzenie pary kluczy: prywatnych i publicznych z użyciem RSA i zapis do pliku

```
private_key.pem
1 AqNEu8OeyE7A63aX7BBAqrjxrGh/m5TPwcxkXN0EN/gg10lILDmu+4hxM1sfR/lFsU1sf9vgkeU8k
  ZvXhp8uumM3ItQ80GT3zSP2LqOIAF9g49favIa00R8ueqIi3Suk+mzkeBS8hSE8mJ7wNi5ebTu9wY
  8FCMk9dQ5ewoYZ2BveUtkr7Jx74tQckYiA0K5AqcEhWLoZ+AJ8fIplXs4EgoY9nRKY2ZSUTTC7D7N
  6nLarsBFfngXhSnUi45xhHkms8jm2GoK+Z52Wi8S71ara5+dS6bxbqvuRB2v/4v8JXBgY6o83KJR
  FcmSgk20M830VvxA2RWnJmf3KlgSBkNR45VBn7u41lxkwHhbPGQTy6JcdgHWHQ8oB23uKIC1reDQv
  g5sGTFkf9azPs/BwCLKmNtu6jdg2y4WlyZXxbv0CWkd2SXbwzEOctHAjij2+Mdro9kLLjQTaLUMeeO
  OlHbJxnSm065m08FUK5hbiE5+4P8jflk8HunEom0sW6Svee86TQXfUbuJvRBMhKEIOEDmu2+J91F
  o+2r3Rd/9cqww7cvy28ySVZRZ2f+L19jOoK7EUSbfMb+2wH9XhZ9hUzX5+4FVCzp+J2uSqEpMFzy
  bvVrXh9X/gWKUj6h4msQUoxjiGPFfEKvHg7sOZFzgdq2+bva+T+n9yudo4FCVDLEA0g8ieLDBjfVO
  DlFON5Up4YrJyR7q8Q0kqSP0yuvrpl7AgJCixTx3vD9rvt8aJev0+6Vtjc0xWBvpOgj4IgWpR/ODo
  Nt0ScSN4M9lkg/zZT4X1arG+u7ea0fftyGfoc5MY8U+qmkggO/4s4SMY4daPxLqAjhlANdlq0bSmc
  bmvo9q7WMsJtx9wzaQM4KxTiAW/Fpm1/TUxZRLJNQqmxDJhbnR2qKAS4VvhkEL1CvXgc74iLeilYn
  wWP+mHpyccsOfjpJoSNG5nBy+TV2UCyPwy4L7y2RxBzSnu0McD85jSu3RVud8+f85Q5ALLKPMFIR
  EY4uHDb520F5hUc+g00JegUTetZfXva5yol2wLkvSp5XI6YQ5nYLQG3FEtcUr6F3jcAllHscBMBjJ
  576gIILb1Y2u+0fzXBfi2Q17IvfOP2j3gxQ5hWBiw0k0e/mosZfUY3TpCLbqXwx1jypoS172QHRIT
  SWos/FRA5bUm880fy/0iileV40TFaalmuEESuV3txmhOoFNrMNAIqOb4x1V1pGWTyf/Zec1lpiwk
  hJWDyv9NK01+oBClYDUOZUyTN21ZRM/sAQho7NmDt6hH8X8vTLyncmPK0ufSTrOYZYOD8Ih6+qYoh
  U7I3s1aC00uS/tbZGhSIJGM05nTzvYvQGFJ5dYftbSPnHVBCAEDK/zhoOfkqLTXwEoDouy9EXKozF
  Cbb37zF7qBnCD1MlEBm9NtoZfi1ST2zD0icf6KTUyqWMO/ObHq10415Ly6l627zMg7eqvk4tbOdo5
  +e6EOMbQG3zQQMmu0nScLf5hsQRJlJKj26n4fXg4VZg/jrwpYlraXSm7TSMn+UHoAcTHWZSWCHDXl
  vOTQtWIG0zHUT8VA8tqov36VuPkSuwxp7jXYXLQUXtKoYPfVHpxGU0cziTacP2UPNVBF/oibr7iOc
  L3Itu8wE+PaPGjgldpuXkzJDL0hm+UxXXj98NC+59nuaLWVmdgRuzj0+vdjSNfFXmrydMtgwIbXCp
  FsogszZregkW4J2B9K3JxeQ+1/U8QQXCDpRZawxOGg4z9bA9CeaErYb5vHFHO5D2fcin6cYWJGbYy
  3g9OgFiCk9wExsw+Ez/PS6/rb0j8gVKp4tSSxpQjEJicFpDnOPW9fnIfuAshfkH2T2PYT0RQirbJa
```

```
public_key.pem
1 -----BEGIN PUBLIC KEY-----
2 MIICIjANBgkqhkiG9w0BAQEFAAOCAg8AMIICCgKCAgEAXhudBAKTauY3+H1ZNYfb
3 0KOnjHVWHvrA5xLffZ3y29z54Ou4P5T0t5TnEecmh9G+gTVCP34ztSiuoEE5/WSX
4 FaiCRQyzVh2PbjfeMzds6JLilervY9DnKQpZYpnu/5PuAv4NXTVB9GXX9VTx60qS
5 BvzskYeR6IvDcQolHMuA8TnNy4yTIQAx10nOKwWdQLHROUKWAtt7qhDKfty4rOtB
6 tqg678uAGNES1ZRamnYEQdc8vL4xHdN7rasFYUQU4xMJp5fPY9sRK/8mk9iRwd
7 e27YNx3e7ceeJ0Wn6CrkxT9oFSXcGq4sEpaN/233B+eW+xYXY9xzIv8lSiCmhqcl
8 udsuNCYgbGkBteNp3Hvef85aCZtxCkOw7oahOvGfvFjPWBdkl64YiOhOXhjNe0qB
9 OtkUWoghJ5goQ6eXqTb70KLmzyLDX/Agn8wjRt9ZbuYG0v8kgOZdNozMssIA5b1O
10 CngY/f6hY1cmRBbclGQenG8WNRoilGSG4Q1eDZ9/3RW6pTpHYVMnKH7JYuVSYeZp
11 jxDdi/I+8+4G7LEgySPka+pismfwwlqewUdQctbkY03WtF+bq90zFRuSYK2e0Gsx
12 wXuXT5gYB2DsyPtukcaLUqLdXNqUb6gMkv07OUCEMj/dN09ov8irgRLJDsn5Hbf+
13 ZEw6nvAswSBMm2b0P+xoV6kCAwEAAQ==
14 -----END PUBLIC KEY-----
```

- Wysyłanie wiadomości i potwierdzanie



- Generacja klucza sesyjnego

```
session key: b'\x11\xbb^\xb5|\xb4\xaa5Q\x923U^\x16}\xc3\x959s\xbe\xa4B\xc3UGJR\xe04\xe1E\x84'
```

1.4 Zdjęcia kodu + komentarze

- Zestawienie połączenia sieciowego poprzez gniazda sieciowe z wykorzystaniem protokołu TCP/IP między procesami

```
self.serverSocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

Rys. Inicjalizowanie gniazda sieciowego z protokołem TCP/IP (parametr: socket.AF_INET)

```
def listen(self):
    while True:
        #print("SERVER: " + str(threading.current_thread().getName()))
        serverSocketName = self.serverSocket.getsockname()
        try:
            # print(str(self.serverSocket))
            (self.clientSocket, clientIpPort) = self.serverSocket.accept()
            self.clientIp = clientIpPort[0]
            self.keyExchange()
            self.logger.log("Established connection with client: " + str(self.clientIp))
            receiverThread = threading.Thread(target=self.receiveMessage, name="Server Receiver", daemon=True)
            receiverThread.start()
        except socket.error:
            self.logger.log("Server Socket: " + str(serverSocketName) + " has been closed")
            break
```

Rys. Funkcja nasłuchująca na połączenie z innym procesem. Przy w przypadku błędu połączenia rzuca i obsługiwany zostaje wyjątek. Po skutecznym nawiązaniu połączenia następuje wymiana kluczy publicznych i sesyjnego, a następnie powoływany zostaje nowy wątek do nasłuchiwanie na wiadomości.

```
def connect(self, serverIp):
    try:
        self.clientSocket.connect((serverIp, self.clientPort))
        self.serverIp = serverIp
        self.keyExchange()
        self.logger.log("Established connection with server: " + serverIp)
        return True
    except socket.error:
        self.logger.log("Couldn't connect to server: " + serverIp)
        return False
```

Rys. Funkcja nawiązująca połączenie z serwerem. Przy w przypadku błędu połączenia rzuca i obsługiwany zostaje wyjątek.

-
- Tworzenie pary kluczy: prywatnych i publicznych z użyciem RSA

```
def generateKeys(self):
    privateKey = rsa.generate_private_key(
        public_exponent=65537,
        key_size=4096
    )

    pemPrivateKey = privateKey.private_bytes(
        encoding=serialization.Encoding.PEM,
        format=serialization.PrivateFormat.PKCS8,
        encryption_algorithm=serialization.NoEncryption()
    )

    self.privateKeyPasswordHash = self.generateHash(self.privateKeyPassword)
    encryptedPemPrivateKey = self.encryptAES(self.privateKeyPasswordHash, pemPrivateKey)

    publicKey = privateKey.public_key()

    pemPublicKey = publicKey.public_bytes(
        encoding=serialization.Encoding.PEM,
        format=serialization.PublicFormat.SubjectPublicKeyInfo
    )
```

Rys. Funkcja tworząca klucze: prywatny i publiczny. Został użyty algorytm RSA z kluczem o rozmiarze 4096 bajtów.

- Zapis pary kluczy do plików, w tym prywatnego w formie zaszyfrowanej używając algorytmu symetrycznego AES w trybie CBC, który jako klucz przyjął hash hasła utworzony poprzez SHA3 w trybie 256 bitowym

```
def generateHash(self, password):
    result = hashlib.sha3_256(password)
    return result.digest()
```

Rys. Generowanie funkcji skrótu wykorzystując algorytm SHA3-256.

```
def encryptAES(self, key, data):
    #self.iv = get_random_bytes(AES.block_size)
    if self.AES_MODE != AES.MODE_ECB:
        self.iv = bytes([0xa3, 0x11, 0xaa, 0xc0, 0x0d, 0xee, 0xf1, 0xff,
                        0x01, 0x03, 0x07, 0x00, 0x25, 0x58, 0x99, 0xc3])
        cipher = AES.new(key, self.AES_MODE, self.iv)
        return b64encode(cipher.encrypt(pad(data, AES.block_size)))
    else:
        cipher = AES.new(key, self.AES_MODE)
        return b64encode(cipher.encrypt(pad(data, AES.block_size)))
```

Rys. Funkcja szyfrująca dane algorytmem symetrycznym AES. Podział na przypadki, gdy: trybem NIE jest ECB (wykorzystanie wektora IV) oraz trybem jest ECB (bez IV).

```
def saveKeysToFile(self, encryptedPrivateKey, publicKey):
    with open(self.pathToPrivate, "w") as privateFile:
        privateFile.write(encryptedPrivateKey.decode())

    with open(self.pathToPublic, "w") as publicFile:
        publicFile.write(publicKey.decode())
```

Rys. Zapisanie do pliku zaszyfrowanego klucza prywatnego oraz klucza publicznego.

```
def readKeysFromFile(self):
    with open(self.pathToPrivate, "rb") as key_file:
        privateKey = serialization.load_pem_private_key(
            self.decryptAES(self.privateKeyPasswordHash, key_file.read()),
            password=None
        )

    with open(self.pathToPublic, "rb") as publicFile:
        publicKey = serialization.load_pem_public_key(publicFile.read())

    return privateKey, publicKey
```

Rys. Odczytanie z pliku zaszyfrowanego klucza prywatnego oraz klucza publicznego.

- Generowanie klucza sesyjnego

```
def generateSessionKey(self):  
    sessionKey = secrets.token_bytes(SESSION_KEY_LENGTH)  
    return sessionKey
```

Rys. Funkcja generująca klucz sesyjny

- Wymiana kluczy publicznych między procesami

```
def keyExchange(self):  
    self.privateKey, self.publicKey = self.encryptor.readKeysFromFile()  
    self.sessionKey = self.encryptor.generateSessionKey()  
    # Send public key to client, receive client's public key and send encrypted session key to client  
    pemPublicKey = self.publicKey.public_bytes(  
        encoding=serialization.Encoding.PEM,  
        format=serialization.PublicFormat.SubjectPublicKeyInfo  
    )  
    self.clientSocket.send(pemPublicKey)  
    pemClientPublicKey = self.clientSocket.recv(MSG_LENGTH)  
    self.clientPublicKey = serialization.load_pem_public_key(pemClientPublicKey)  
  
    encryptedSessionKey = self.clientPublicKey.encrypt(  
        self.sessionKey,  
        padding.OAEP(  
            mgf=padding.MGF1(algorithm=hashes.SHA256()),  
            algorithm=hashes.SHA256(),  
            label=None  
        )  
    )  
    self.clientSocket.send(encryptedSessionKey)  
  
    print(f'Client public key: {self.clientPublicKey}')  
    print(f'Session key: {self.sessionKey}')  
    print(f'Encrypted session key: {encryptedSessionKey}')  
    print(f'Encrypted session key: {encryptedSessionKey}')
```

Rys. Serwerowa funkcja wymieniająca klucze między procesami. Na początku wczytywane z plików zostają klucze: prywatny i publiczny. Następnie generowany jest klucz sesyjny. Następnie serwer wysyła do klienta swój klucz publiczny i vice versa. Następnie serwer szyfruje klucz sesyjny kluczem publicznym klienta i mu go wysyła.

- Wymiana kluczy między procesami

```
def keyExchange(self):
    # Send public key to server and receive server's public key and encrypted session key
    self.privateKey, self.publicKey = self.encryptor.readKeysFromFile()
    pemPublicKey = self.publicKey.public_bytes(
        encoding=serialization.Encoding.PEM,
        format=serialization.PublicFormat.SubjectPublicKeyInfo
    )

    try:
        self.clientSocket.send(pemPublicKey)
        pemServerPublicKey = self.clientSocket.recv(MSG_LENGTH)
        self.serverPublicKey = serialization.load_pem_public_key(pemServerPublicKey)
        encryptedSessionKey = self.clientSocket.recv(MSG_LENGTH)
        self.sessionKey = self.privateKey.decrypt(
            encryptedSessionKey,
            padding.OAEP(
                mgf=padding.MGF1(algorithm=hashes.SHA256()),
                algorithm=hashes.SHA256(),
                label=None
            )
        )
    except socket.error:
        if self.serverIp is not None:
            self.logger.log("Server " + self.serverIp + " has been disconnected!")
        else:
            self.logger.log("You are not allowed to exchange the public keys. Connect to the server!")

    print(f'Server public key: {self.serverPublicKey}')
    print(f'Encrypted Session key: {encryptedSessionKey}')
    print(f'Decrypted Session key: {self.sessionKey}')
```

Rys. Klientka funkcja wymieniająca klucze. Wysyła swój klucz publiczny serwerowi i otrzymuje jego klucz publiczny oraz zaszyfrowany klucz sesyjny, który odszyfrowuje swoim kluczem prywatnym.

- Wysyłanie i odbieranie wiadomości oraz wysłanie potwierdzenia ich odbioru przez serwer

```
def sendMessage(self, message):
    # print(message)
    try:
        self.clientSocket.send(self.encryptMessage(message))
        self.clientSocket.settimeout(1.0)
        ackMessage = self.clientSocket.recv(MSG_LENGTH).decode()
        if ackMessage is not None:
            self.logger.log(ackMessage)
    except socket.error:
        if self.serverIp is not None:
            self.logger.log("Server " + self.serverIp + " has been disconnected!")
        else:
            self.logger.log("You are not allowed to send a message. Connect to the server!")
```

Rys. Funkcja wysyłająca zaszyfrowaną wiadomość do serwera. Następnie odbierana jest wiadomość potwierdzająca odbiór.

```
def receiveMessage(self):
    while True:
        # print("SERVER: " + str(threading.current_thread().getName()))
        try:
            (readyToRead, readyToWrite, connectionError) = select.select([self.clientSocket], [], [])
            encryptedMessage = self.clientSocket.recv(MSG_LENGTH)
            message = self.decryptMessage(encryptedMessage)
            if len(message) > 0:
                self.logger.log(message)
                self.clientSocket.send(ACK_MESSAGE.encode())
            elif len(message) == 0 and len(encryptedMessage) > 0:
                self.logger.log("Message decryption failed! Check if you are using a good encryption mode!")
                self.clientSocket.send(ACK_ERROR_MESSAGE.encode())
            elif len(message) == 0 and len(encryptedMessage) == 0:
                self.logger.log("Client " + self.clientIp + " has been disconnected!")
                break
        except select.error:
            self.logger.log("Client " + self.clientIp + " has been disconnected xd!")
            self.clientSocket.close()
            break
```

Rys. Funkcja odbierająca wiadomość. Odebrana wiadomość jest następnie odszyfrowywana. Po poprawnym odbiorze, wysłana zostaje wiadomość potwierdzająca odbiór.

- Szyfrowanie i odszyfrowywanie wiadomości algorytmem symetrycznym AES z kluczem sesyjnym

```
def encryptMessage(self, message):
    print(f'session key: {self.sessionKey}')
    encryptedAESMessage = self.encryptor.encryptAES(self.sessionKey, message.encode())
    print(f'message: {message}')
    print(f'Encrypted message: {encryptedAESMessage}\n')
    return encryptedAESMessage
```

Rys. Funkcja obsługująca szyfrowanie wiadomości. Wykorzystuje funkcję szyfrującą (opisaną już wcześniej).

```
def decryptAES(self, key, data):
    if self.AES_MODE != AES.MODE_ECB:
        cipher = AES.new(key, self.AES_MODE, self.iv)
    else:
        cipher = AES.new(key, self.AES_MODE)
    data = b64decode(data)
    try:
        result = unpad(cipher.decrypt(data), AES.block_size)
    except ValueError:
        result = bytes()

    return result
```

Rys. Funkcja odszyfrowująca wiadomość. W zależności od trybu szyfrowania (ECB albo którykolwiek inny) tworzony jest odpowiedni szyfr (z wektorem IV lub nie).

- Możliwość wyboru trybu szyfrowania algorytmu AES: CBC, ECB, CFB, OFB

```
def switchEncryptionMode(self, AES_MODE):
    if AES_MODE == "CBC":
        self.AES_MODE = AES.MODE_CBC
    elif AES_MODE == "ECB":
        self.AES_MODE = AES.MODE_ECB
    elif AES_MODE == "CFB":
        self.AES_MODE = AES.MODE_CFB
    elif AES_MODE == "OFB":
        self.AES_MODE = AES.MODE_OFB
```

Rys. Funkcja obsługująca wybór trybu szyfrowania AES

1.5 Plany na przyszłość

W przyszłości planujemy zaimplementować następujące funkcjonalności:

- Wysyłanie plików
- Szyfrowanie i deszyfrowanie plików
- Pasek postępu wysyłania
- Możliwość wysyłania dużych plików
- Automatyczne dopasowanie trybu odszyfrowywania
- Zachowanie rozszerzenia pliku po jego odszyfrowaniu

2. Projekt – termin kontrolny

2.1 Uwagi wstępne

W tej części nie będziemy opisywać już funkcjonalności zaimplementowanych do połowy semestru, ponieważ są one już opisane w pierwszej części raportu. Jedynie opiszemy i wrzucimy zrzuty ekranu nowozaimplementowanych funkcjonalności oraz tych z pierwszej połowy semestru, które uległy modyfikacjom.

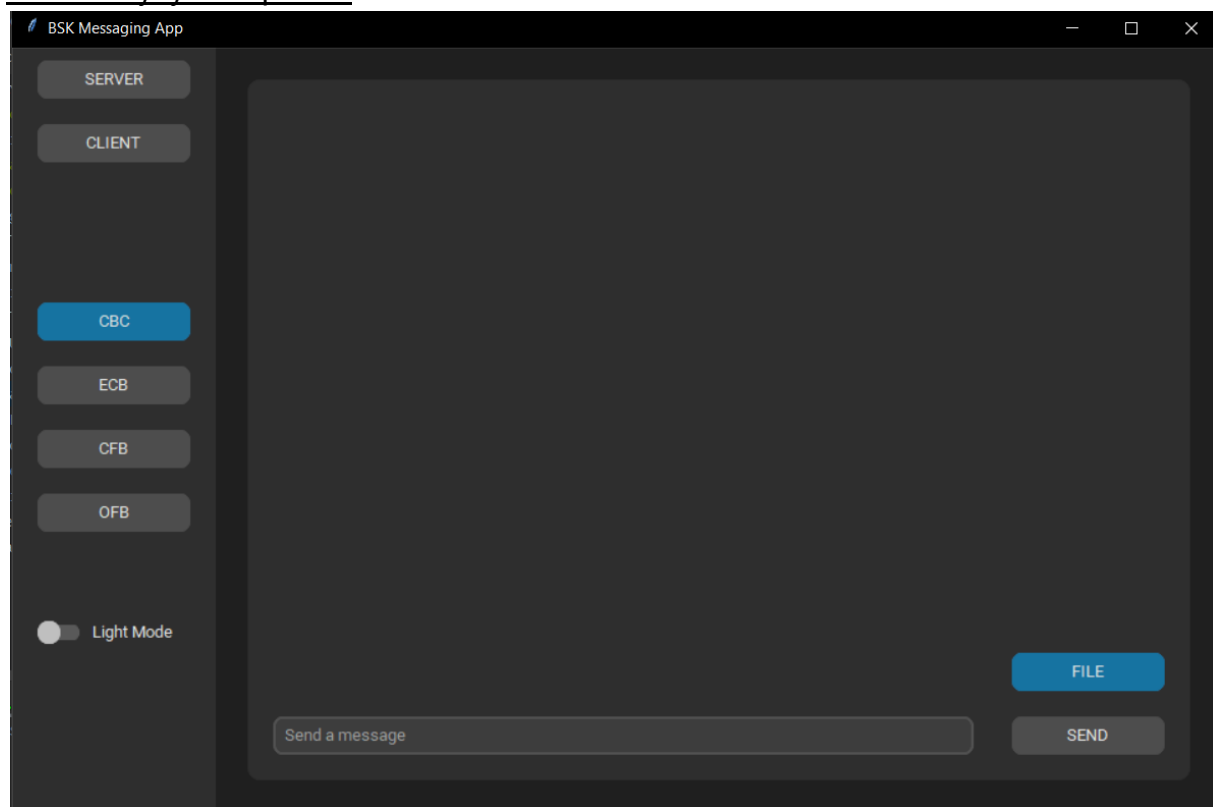
2.2 Nowe funkcjonalności

W drugiej części semestru dodaliśmy poniższe funkcjonalności do projektu:

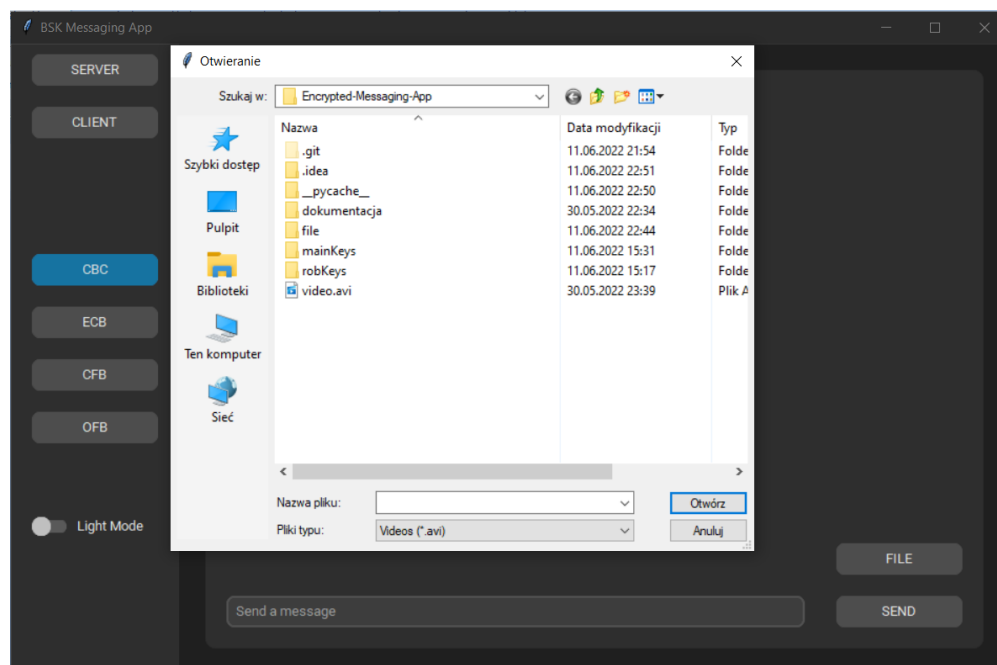
- Wysyłanie plików
- Wybór rozszerzenia plików w GUI (*.txt, *.png, *.pdf, *.avi)
- Pasek postępu wysyłania
- Zachowywanie rozszerzenia plików
- Poprawne odszyfrowywanie małych i dużych plików
- Szyfrowanie i wysyłanie plików ponad 500 MB
- Zachowanie rozszerzenia pliku po jego odszyfrowaniu

2.3 Zdjęcia programu

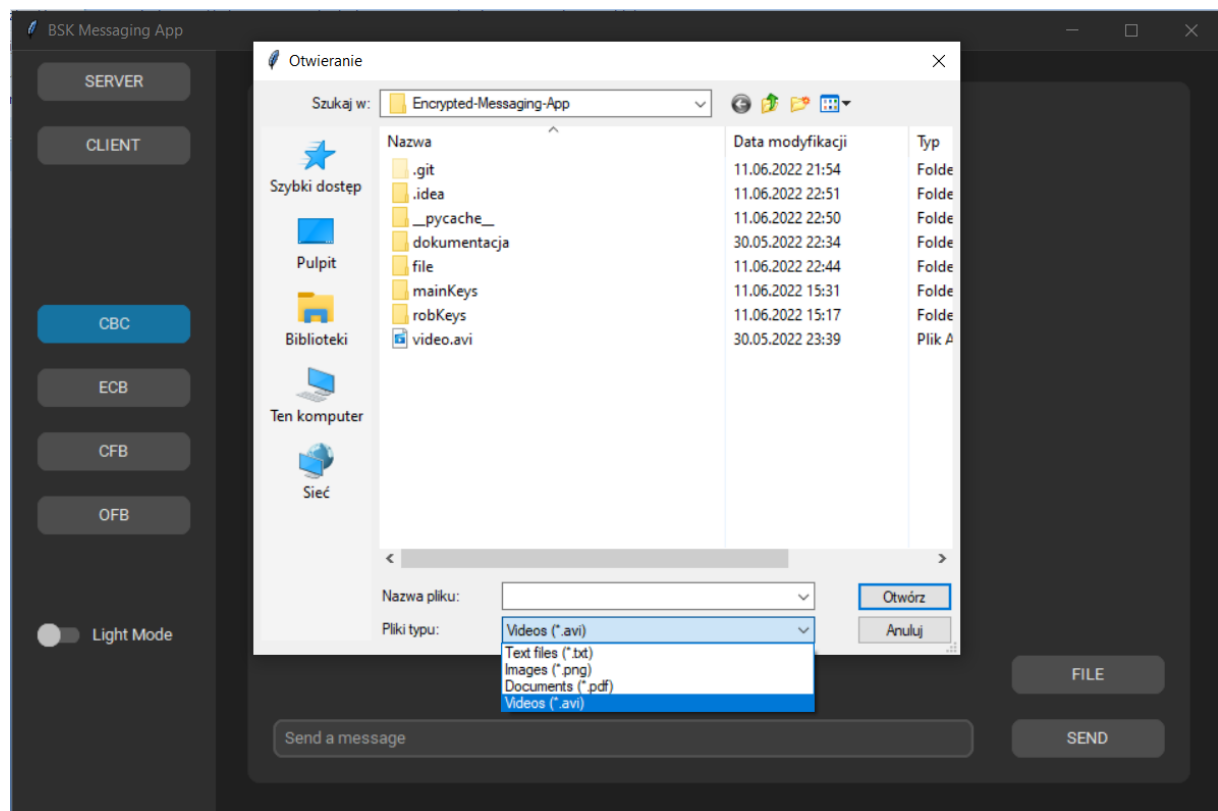
- Proces wysyłania plików



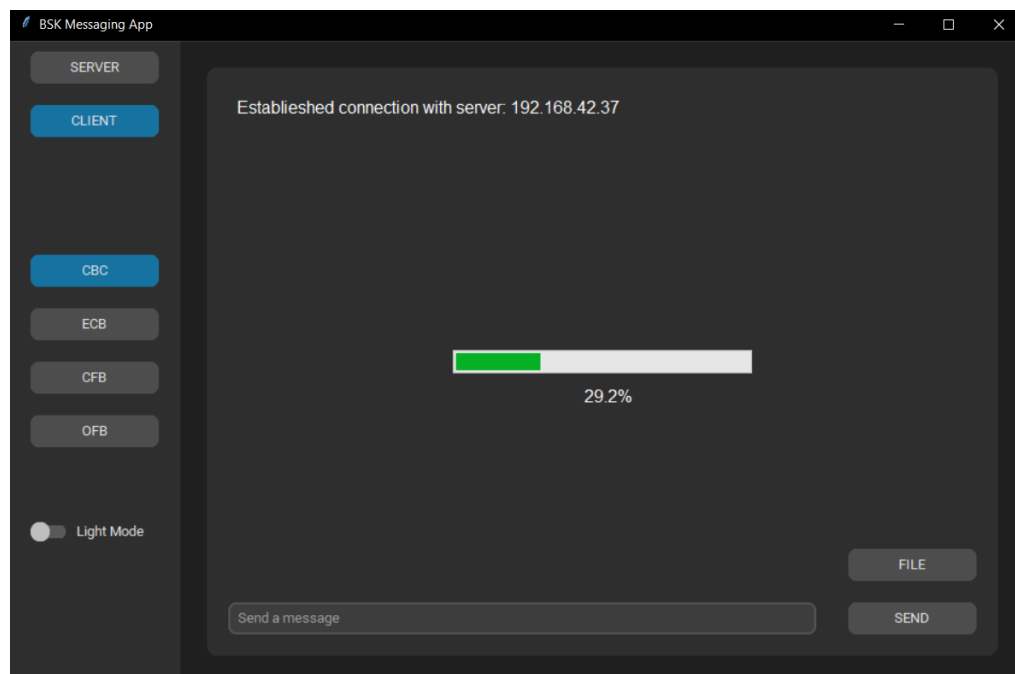
Rys. Dodany przycisk 'FILE' umożliwiający załadowanie pliku do programu



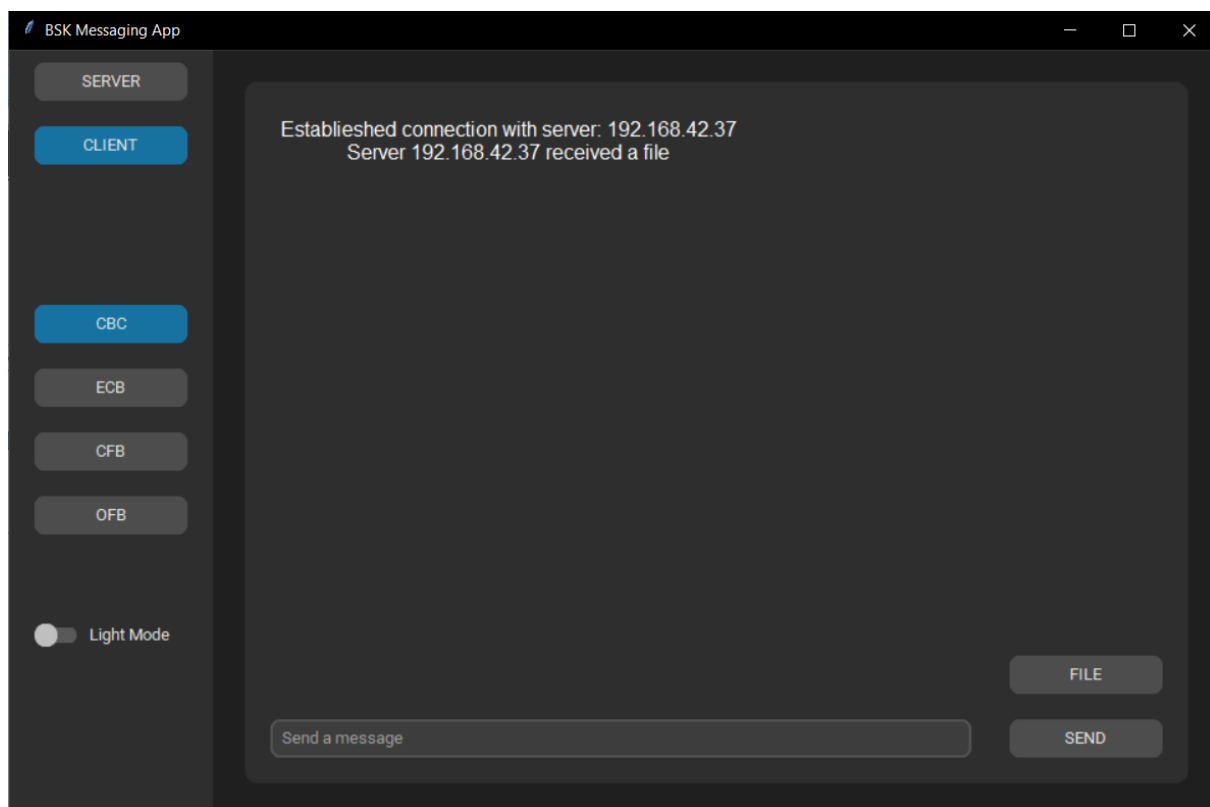
Rys. Wyświetlenie eksploratora plików





*Rys. Możliwość wyboru jednego z czterech rozszerzeń:
*.txt, *.png, *.pdf, *.avi*





Rys. Pasek postępu podczas wysyłania pliku





Rys. Potwierdzenie odebrania pliku przez drugą aplikację

 video.avi	30.05.2022 23:39	Plik AVI	726 KB
 video_decrypted.avi	11.06.2022 23:00	Plik AVI	726 KB



*Rys. Zapisany plik o takim samym rozmiarze i rozszerzeniu *.avi*

 DEFINICJE.pdf	14.01.2022 21:07	Adobe Acrobat D...	472 KB
 DEFINICJE_decrypted.pdf	11.06.2022 23:08	Adobe Acrobat D...	472 KB

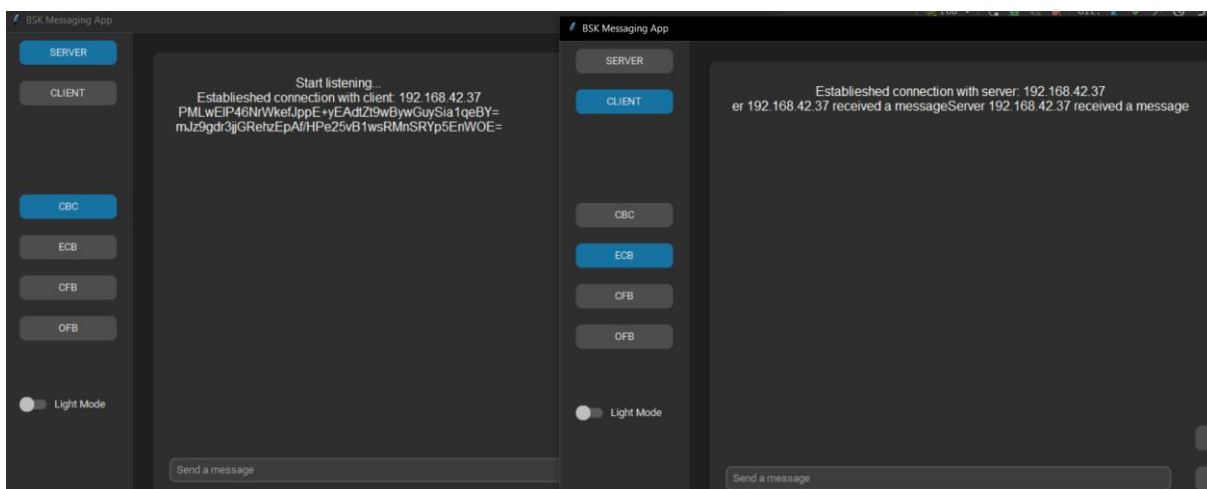
*Rys. Zapisany plik o takim samym rozmiarze i rozszerzeniu *.pdf*

 loremlpsum.txt	06.06.2022 18:11	Dokument tekstowy	78 KB
 loremlpsum_decrypted.txt	11.06.2022 23:11	Dokument tekstowy	78 KB

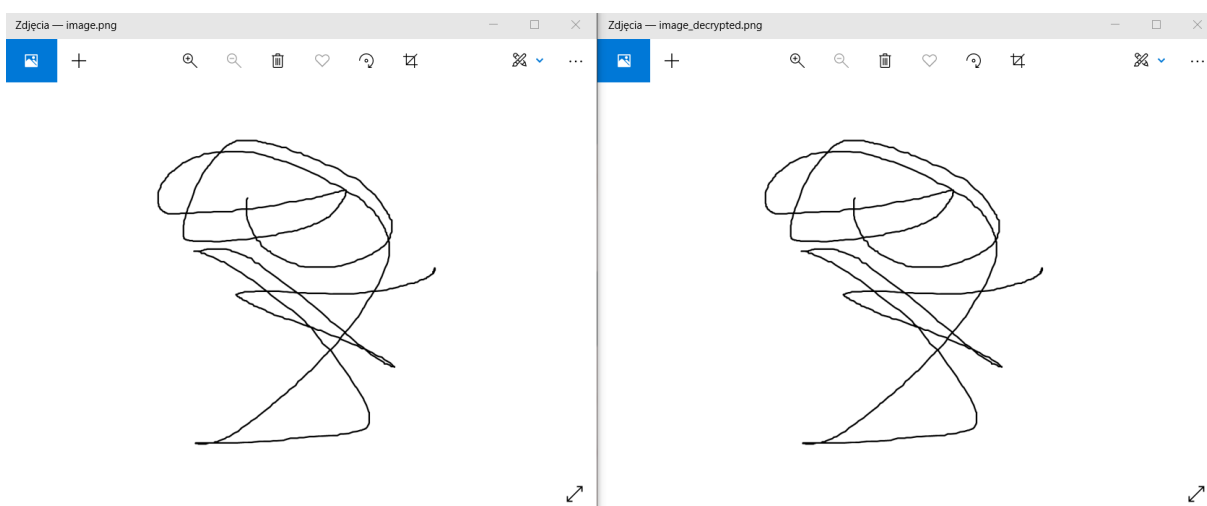
*Rys. Zapisany plik o takim samym rozmiarze i rozszerzeniu *.txt*

 image.png	30.05.2022 23:16	Plik PNG	1 511 KB
 image_decrypted.png	11.06.2022 23:13	Plik PNG	1 511 KB

*Rys. Zapisany plik o takim samym rozmiarze i rozszerzeniu *.png*



Rys. Rezultat w przypadku wysłania i odebrania plików w przypadku błędnych kluczy/trybów szyfrowania



Rys. Plik początkowy przed zaszyfrowaniem i wysłaniem i odebrany po odszyfrowaniu

test.txt	11.06.2022 22:18	Dokument tekstowy	512 000 KB
test_decrypted.txt	12.06.2022 00:45	Dokument tekstowy	512 000 KB

Rys. Poprawnie odszyfrowany plik o rozmiarze przekraczającym 500 MB.

2.4 Zdjęcia kodu + komentarze

```
class FileHandler:
    fileName = None
    content = None
    extension = None

    def openFileDialog(self):
        file = filedialog.askopenfile(mode='rb', filetypes=[('Text files', '*.txt'),
                                                            ('Images', '*.png'),
                                                            ('Documents', '*.pdf'),
                                                            ('Videos', '*.avi')])
        self.readFromFile(file)

    def readFromFile(self, file):
        if file is not None:
            self.content = file.read()
            filePath = file.name
            self.fileName = filePath.rsplit('/', 1)[-1]
            print(self.fileName)

    def openFile(self, fileName):
        file = open(fileName, 'ab')
        return file

    def saveToFile(self, message, file):
        file.write(message)
        time.sleep(0.000001)

    def closeFile(self, file):
        file.close()
```

Rys. Klasa FileHandler odpowiadająca za otwieranie, zamykanie pliku, odczytywanie z i zapisywanie do pliku bajtów.

```

def sendFile(self, file, progressBar):
    try:
        self.clientSocket.send("file_begin".encode())
        self.clientSocket.send(self.encryptMessage(self.fileHandler.fileName.encode()))
        bytesInFile = len(self.fileHandler.content)
        for i in range(0, bytesInFile, MSG_FILE_LENGTH):
            testMessage = self.fileHandler.content[i:i + MSG_FILE_LENGTH]
            self.clientSocket.send(self.encryptMessage(testMessage))
            nextBytes = self.clientSocket.recv(4)
            time.sleep(0.01)
            value = (i / bytesInFile) * 100
            self.updateProgressBar(progressBar, value)

        self.updateProgressBar(progressBar, 100)
        self.clientSocket.send(self.encryptMessage("file_end".encode()))
        time.sleep(0.01)

        ackMessage = self.clientSocket.recv(MSG_LENGTH).decode()
        if ackMessage is not None:
            self.logger.log(ackMessage)

    except socket.error:
        if self.serverIp is not None:
            self.logger.log("Server " + self.serverIp + " has been disconnected!")
        else:
            self.logger.log("You are not allowed to send a message. Connect to the server!")

    return None

```

Rys. Funkcja wysyłania pliku do drugiej aplikacji. Na początku wysyłany jest nagłówek z informacją, że wysyłany jest plik, następnie nazwa pliku wraz z rozszerzeniem. Później szyfruje i wysyła zawartość pliku podzieloną na wiele części, o długościach określonych zmienną zapisaną w pliku globals.py. Po wysłaniu każdej części, aktualizuje pasek postępu. Na samym końcu wysyła informację, że wysłał już wszystkie części i otrzymuje zwrotną informację od drugiej aplikacji, że pomyślnie otrzymał wszystkie części. Celem podziału pliku na części i wysyłania go we fragmentach była możliwość obsługi dużych plików.

```
def receiveController(self):
    while True:
        try:
            (readyToRead, readyToWrite, connectionError) = select.select([self.clientSocket], [], [])
            encryptedMessage = self.clientSocket.recv(MSG_LENGTH)
            message = self.decryptMessage(encryptedMessage).decode()
            if len(message) > 0:
                if encryptedMessage.decode() == "file_begin":
                    self.receiveFile()
                    self.clientSocket.send(ACK_MESSAGE_FILE.encode())
                else:
                    self.logger.log(message)
                    self.clientSocket.send(ACK_MESSAGE.encode())
            elif len(message) == 0 and len(encryptedMessage) > 0:
                self.logger.log(message)
                self.clientSocket.send(ACK_MESSAGE.encode())
            elif len(message) == 0 and len(encryptedMessage) == 0:
                self.logger.log("Client " + self.clientIp + " has been disconnected!")
                break
        except select.error:
            self.logger.log("Client " + self.clientIp + " has been disconnected!")
            self.clientSocket.close()
            break
```

Rys. Funkcja będąca kontrolerem otrzymywanych wiadomości. Funkcja ta zastąpiła funkcję receiveMessage() opisaną w raporcie kontrolnym ze względu na rozszerzenie funkcjonalności o wysyłanie plików, w związku z czym aplikacja musi wykryć, czy ma być odbierany plik czy wiadomość tekstowa. W przypadku odbierania pliku (odszyfrowany nagłówek o wartości: „file_begin”), wywoływana jest dedykowana funkcja do odbioru pliku, a w przeciwnym razie funkcja samodzielnie obsługuje odbiór wiadomości, czyli wypisuje odszyfrowaną wiadomość na ekran. W każdym z tych dwóch przypadków wysyłana jest odpowiednia wiadomość potwierdzająca do drugiej aplikacji.

```
def receiveFile(self):
    (readyToRead, readyToWrite, connectionError) = select.select([self.clientSocket], [], [])

    encryptedFileName = self.clientSocket.recv(MSG_LENGTH)
    fileName = self.decryptMessage(encryptedFileName).decode()
    fileName = f"{fileName.rsplit('.', 1)[0]}_decrypted.{fileName.rsplit('.', 1)[-1]}"
    file = self.fileHandler.openFile(fileName)
    threadDelegated = False
    fileSaver = None
    while True:
        encryptedMessage = self.clientSocket.recv(MSG_LENGTH)
        message = self.decryptMessage(encryptedMessage)

        if message == "file_end".encode():
            break

        if threadDelegated:
            fileSaver.join()

        fileSaver = threading.Thread(target=self.fileHandler.saveToFile, args=(message, file),
                                     name="File saver", daemon=True)
        fileSaver.start()
        if not threadDelegated:
            threadDelegated = True

        self.clientSocket.send("next".encode())

    self.fileHandler.closeFile(file)
```

Rys. Funkcja do odbierania plików. Na początku odbierana jest nazwa pliku wraz z rozszerzeniem. Następnie tworzony i otwierany jest plik o takiej samym rozszerzeniu i nazwie, ale z dodanym przyrostkiem „_decrypted”. Potem odbierane są części pliku. Następnie są one odszyfrowywane. Później delegowany jest nowy wątek do zapisu danej części do pliku. W kolejnej iteracji ponownie odbiera i odszyfrowuje wiadomości, ale deleguje nowy wątek dopiero wtedy, gdy ten wcześniejszy już się zakończył (żeby dwa wątki nie pisały jednocześnie do pliku). Wysyłany jest też komunikat do drugiej aplikacji, że może wysłać kolejną część pliku. Celem podziału pliku na części i odbierania go we fragmentach była możliwość obsługi dużych plików.

2.5 Uwagi końcowe

Pliki były szyfrowane i odszyfrowywane z użyciem symetrycznego klucza sesyjnego. Klucz sesyjny był ustalany z użyciem kryptografii asymetrycznej. Jedna aplikacja generowała klucz symetryczny, szyfrowała go kluczem publicznym drugiej aplikacji, która odszyfrowywała go swoim kluczem prywatnym. Klucz symetryczny był generowany podczas nawiązywania połączenia i jest ważny do momentu zerwania tego połączenia.

Pliki były otwierane z pliku, szyfrowane, wysyłane, odbierane, deszyfrowane i zapisywane do pliku w formie bajtowej. Z tego tytułu, możliwe było przesyłanie plików o różnych rozszerzeniach (zdjęcia, dokumenty, tekstowe, video) w ten sam sposób, z użyciem tych samych funkcji. Ponadto, szyfrowanie i deszyfrowanie następowało w identyczny sposób, niezależnie od tego, czy mieliśmy do czynienia z wiadomością przesyłaną z poziomu aplikacji czy też z plikiem – również ze względu na to, że obie formy były w postaci bajtów. Funkcje szyfrowania i deszyfrowania zostały opisane w kontrolnej części raportu z połowy semestru.

3. Bibliografia

- [1] <https://docs.python.org/3/library/socket.html>
- [2] <https://realpython.com/python-sockets/>
- [3] <https://www.geeksforgeeks.org/socket-programming-python/>
- [4] <https://pycryptodome.readthedocs.io/en/latest/>
- [5] <https://pycryptodome.readthedocs.io/en/latest/src/cipher/cipher.html>
- [6] <https://docs.python.org/3/library/tkinter.html>
- [7] <https://www.youtube.com/watch?v=Ew9dsDDy7pk>
- [8] <https://docs.python.org/3/library/threading.html>
- [9] <https://www.windows-commandline.com/how-to-create-large-dummy-file/>
- [10] https://notebooks.githubusercontent.com/view/ipynb?browser=chrome&color_mode=auto&commit=a9fc800ae1916c28a0de3cf9e5a9682ac6e130a6&device=unknown&enc_url=68747470733a2f2f7261772e67697468756275736572636f6e74656e742e636f6d2f616d6972616c69732f707974686f6e2d63727970746f2d7475746f7269616c2f613966633830306165313931366332386130646533636639653561393638326163366531333061362f63727970746f2e6970796e62&logged_in=false&nwo=amiralis%2Fpython-crypto-tutorial&path=crypto.ipynb&platform=android&repository_id=20068229&repository_type=Repository&version=99
- [11] <https://programtalk.com/python-examples/cryptography.hazmat.primitives.ciphers.Cipher/?ipage=2>