
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

FACULTY OF ELECTRICAL ENGINEERING

OBOR INFORMAČNÍ BEZPEČNOST

INFORMATION SECURITY


# SDÍLENÍ DŮVĚRNÝCH SOUBORŮ

SEMESTRÁLNÍ PROJEKT 2022

SEMESTRAL PROJECT 2022


AUTOŘI PRÁCE

AUTHORS

Samuel Kopecký (211799)

Petra Kajánková (211550)

Klára Blahušová (211781)

Erik Kuna (211800)

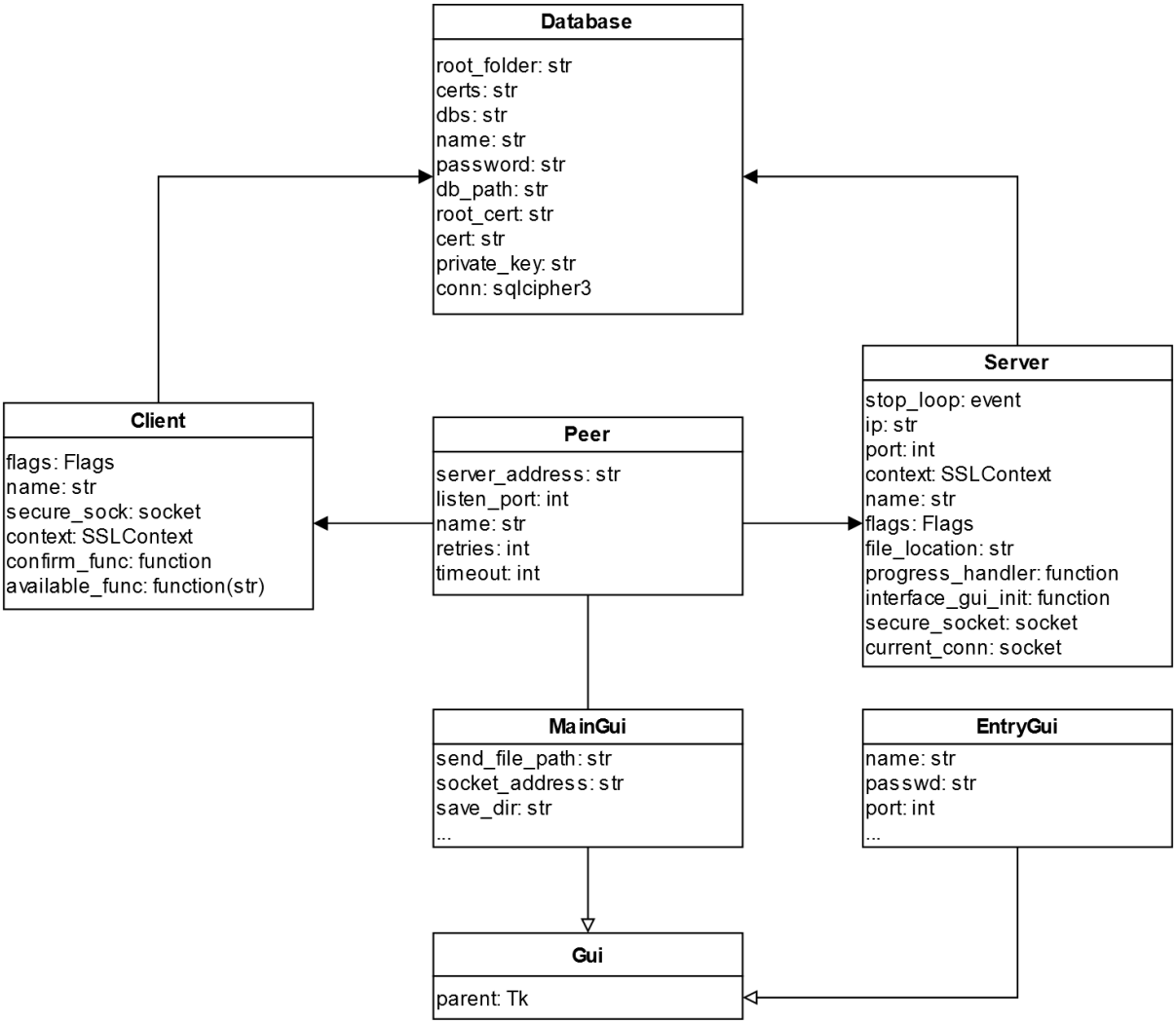
# OBSAH

# SEZNAM OBRÁZKŮ

# ÚVOD

Semestrální projekt se věnuje aplikaci na sdílení důvěrných souborů do předmětu Kryptografie (MPC-KRY). Úkolem je realizovat zabezpečený přenos souborů po síti mezi dvěma a více uživateli. Systém bude postaven na komunikace Peer-to-peer bez možnosti použití serveru jako prostředníka. Aplikace bude realizovat přenos souborů minimálně ve dvou základních režimech a to real-time a odložený přenos. Výstupem projektu bude aplikace s grafickým rozhraním, které bude umožňovat nahrát a odeslat šifrované soubory vybraným uživatelům přes veřejný kanál. Uživatelé a jejich certifikáty budou v rámci aplikace uloženy do databáze.

# VÝVOJOVÝ DIAGRAM




*Obrázek 1: Vývojový diagram*

# POPIS KÓDU

Program je implementován v jazyku Python. V řešení jsou využity dvě knihovny, které nejsou součástí native knihoven Pythonu: Tkinter a Sqlcipher3. Pro podrobnější zobrazení a popis kódu byla vytvořena anglická dokumentace prostřednictvím shpinx a celý projekt je zveřejněn na github.com. Následující rozbor slouží ke krátkému představení dílčích tříd a zajímavých částí kódu.

## Databáze

Je vytvořena zabezpečená databáze s přístupovým heslem. Uživatel spustí program, zadá heslo a k danému heslu se vytvoří databáze. Sqlcipher3 knihovna slouží k zabezpečení dané databáze, která je zašifrovaná pomocí AES 256 klíčem vygenerovaným ze vstupního hesla, které uživatel zadal. Databáze obsahuje položky hostname, IP adresa, port společně s cestami k certifikátům a soukromým klíčům uživatelů. Soukromý klíč je zašifrovaný pomocí hesla.

`DEF CREATE_DATABASE(SELF)` = vytvoří databázi pro uživatele, spustí script na vytvoření soukromého klíče, pokud takový klíč existuje, tak se přihlásí daný uživatel. Existující klíč není přepsán.

```
if not os.path.exists(self.dbs):
    os.mkdir(self.dbs)
self.connect_to_db()
self.create_tables()
if not os.path.exists(self.cert) or not os.path.exists(self.cert) or not os.path.exists(self.private_key):
    self.create_certs()
    self.insert_app(self.root_cert, self.private_key, self.cert)
    return
if len(self.get_table(self.app)) == 0:
    self.insert_app(self.root_cert, self.private_key, self.cert)
```

*Obrázek 2: funkce vytvářející databázi*

`DEF INSERT_USER (SELF, ADDR)` = funkce na vložení uživatele do databáze, pokud se v ní již stejný uživatel nenachází. Parametry addr jsou IP adresa a port uživatele.

```
sql = f'SELECT * FROM users WHERE addr="{addr}"'
result = self.conn.execute(sql).fetchall()
if len(result) != 0 and result[0][1] == addr:
    return
sql = f'INSERT INTO users (addr) VALUES ("{addr}")'
self.conn.execute(sql)
self.conn.commit()
```

*Obrázek 3: funkce vkládání uživatele do databáze*

`DEF INSERT_APP (SELF, ROOT_CERT, PRIVATE_KEY, CERT)` = funkce vloží soukromý klíč a certifikát do databáze. Obsahuje parametry kořenového certifikátu, soukromého klíče uživatele a samotný certifikát uživatele.

```
print("Inserting to app table")
sql = "INSERT INTO app (root_cert, private_key, cert) VALUES (?, ?, ?)"
val = (root_cert, private_key, cert)
self.conn.execute(sql, val)
self.conn.commit()
```

*Obrázek 4: funkce vkládání klíče do databáze*

Obecné shrnutí kódu databáze v angličtině: source.db package.

Přesné znění kódu s popisem v angličtině: MPC-KRY/source/db/Database.py.

## Klient

Třída klient se stará o posílání souboru a Heartbeat komunikaci. Zaopatřuje bezpečnost prostřednictvím TLS protokolu. Klient vyžaduje od druhé strany certifikát před navázáním spojení. Heartbeat komunikace je používána ke zjištění, jestli je protější klient dostupný. Klient pošle zprávu označenou jako HEARTBEAT, pokud bude klientovi tato zpráva na vrácena, může probíhat spojení.

`DEF INIT_SOCK(SELF)` = funkce inicializuje proces pro SSL. Nahrává certifikáty a rozhoduje, jestli během komunikace budou certifikáty vyžadovány.

```
self.context = ssl.create_default_context()
table = self.db.get_table(Database.app)[0]
self.context.load_cert_chain(table[3], table[2], password=self.passwd)
self.context.load_verify_locations(table[1])
self.context.check_hostname = False
self.context.verify_mode = ssl.CERT_REQUIRED
```

*Obrázek 5: funkce incializující zabezpečení*

`DEF SEND_HEARTBEAT(SELF, HOSTNAME, PORT, TIMEOUT)` = funkce posílá Heartbeat zprávy protějšímu klientovi a kontroluje, jestli dané zprávy obdržel. Parametr hostname je IP adresa příjemce, port je portem příjemce a timeout je čas za který bude Heartbeat ukončen.

```
try:
    self.connect(hostname, port, timeout)
except ConnectionRefusedError:
    print("Connection refused")
    return False
self.secure_sock.send(self.flags.HEARTBEAT)
print("Sending heartbeat message")
try:
    received = self.secure_sock.recv(2048)
except socket.timeout:
    print("Connection timeout")
    return False
if received == self.flags.HEARTBEAT:
    print("Received heartbeat back")
    self.available_func(True)
    self.close_conn()
    return True
```

*Obrázek 6: funkce starající se o dostupnost druhého klienta*

Obecné shrnutí třídy Klient v angličtině: source.peer package.

Přesné znění kódu s popisem v angličtině: MPC-KRY/source/peer/Client.py.

## Peer

Komunikační koncový bod aplikace. Je tvořen z tříd Klient a Server. Třída je zodpovědná za zpracování zpráv Heartbeat, které kontrolují dostupnost obou stran. Pokud je druhý peer nedostupný, je vytvořen proces na pozadí, který se pokouší odeslat soubor v zadaném intervalu, dokud neuplyne určitý čas zadaný uživatelem. Soubor, který je odeslán, je zašifrován, dokud není přijímající připraven.

`DEF SEND_FILE(SELF, HOSTNAME, PORT, FILE_PATH, GUI_UPDATE, LIST_UPDATE)` = funkce, která rozhoduje jestli soubor bude poslán nebo jestli bude pozdržen. Parametry hostname je IP adresa příjemce, port je portem příjemce, file_path je cesta k souboru, který chceme poslat a potom dva parametry aktualizující grafické rozhraní.

```
self.client.db.insert_user(f'{hostname}:{port}')
list_update()
if not self.is_alive(hostname, port):
    self.client.available_func(False)
    print("Creating a subprocess for sending a file")
    print(self.timer_timeout)
    command = f'{os.path.abspath("app.py")} -bg {hostname} {str(port)} {file_path} {self.name} {self.passwd} {self.timer_timeout}'
    subprocess.Popen(command, shell=True)
    # Stop sending file in this process
    return
else:
    self.client.available_func(True)
gui_update()
self.client.connect(hostname, port)
file_bytes, file_name = self.file_open_name(file_path)
self.client.send_file(file_bytes, file_name)
```

*Obrázek 7: funkce posílání souboru skrz peer*

`DEF BACKGROUND_SEND` = funkce, která se stará o opětovné zasílání souboru, pokud je druhý peer nedostupný. Data jsou dočasně uložena do šifrovaného souboru.

```
start = int(time.time())
file_bytes, file_name = self.file_open_name(file_path)

# Generate key and IV
aes_key, aes_iv = os.urandom(32), os.urandom(16)
cipher = Cipher(algorithms.AES(aes_key), modes.CBC(aes_iv), default_backend())

# Make dir for temp encrypted files
if not os.path.exists(self.encrypted_files):
    os.mkdir(self.encrypted_files)

# Encrypt and save
encrypted_file_path = f'{self.encrypted_files}/{file_name}_{start}'
encrypted_file = open(encrypted_file_path, 'wb')
encrypted_file.write(self.aes_encrypt(cipher, file_bytes))
encrypted_file.close()
del file_bytes

# Wait for other peer
while not self.client.send_heartbeat(hostname, port, self.timeout):
    if int(time.time() - start) >= self.timer_timeout:
        print("timed out")
        os.remove(encrypted_file_path)
        exit(0)
    sleep(loop_interval)

# Decrypt and send
encrypted_file = open(encrypted_file_path, 'rb')
file_bytes = self.aes_decrypt(cipher, encrypted_file.read())
encrypted_file.close()
os.remove(encrypted_file_path)
self.client.connect(hostname, port)
self.client.send_file(file_bytes, file_name)
exit(0)
```

*Obrázek 8: funkce uložení a opětovného posílání souboru při nedostupnosti klienta*

Obecné shrnutí třídy Peer v angličtině: source.peer package.

Přesné znění kódu s popisem v angličtině: MPC-KRY/source/peer/Peer.py.

## Server

Nejedná se o server zprostředkovávající komunikaci. Jedná se o název třídy, která se na rozdíl od klienta, stará o přijímání souboru. Podobně jako klient zaopatřuje bezpečnost prostřednictvím TLS protokolu a vyžaduje od druhé strany certifikát před navázáním spojení.

`DEF RECEIVE_BODY(SELF, FILE_PATH, CONN, FILE_LEN, FILE_NAME, FILE_DATA, DATA_END)` = funkce, která se stará o přijímání souborů, množství přijatých dat je vyjádřeno v %. Pokračuje v přijímání dat, dokud nepřijde zpráva s informací o data_end. Funkce se stará o uložení souborů. Parametr file_path je cesta, kam se má posílaný soubor uložit. Následují parametry specifikující soubor jako jeho délka, jméno souboru a data.

```
raw_data = bytes()
original_len = file_len
yielded_value = 0
file = open(f"{file_path}/{file_name.decode('UTF-8')}", 'wb')
file.write(file_data)
while not data_end and not self.is_data_end(raw_data):
    try:
        raw_data = conn.recv(4096)
    except (ConnectionResetError, TimeoutError):
        print("Connection error")
        break
    if self.is_data_end(raw_data):
        # Write everything but the DATA_END flag
        file.write(raw_data[:-len(self.flags.DATA_END)])
        break
    file.write(raw_data)
    file_len -= len(raw_data)
    received = 100 - round(file_len / original_len * 100)
    if yielded_value != received and received != 100:
        yielded_value = received
        yield received
print("Done receiving file, sending FIN")
yield 100
file.close()
conn.send(self.flags.FIN)
```

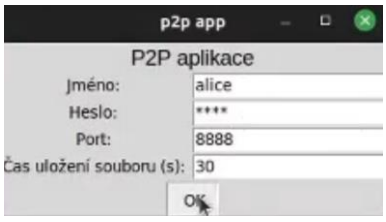
*Obrázek 9: funkce která se stará o přijímání souboru*

Obecné shrnutí třídy Server v angličtině: source.peer package.

Přesné znění kódu s popisem v angličtině: MPC-KRY/source/peer/Server.py.

## GUI

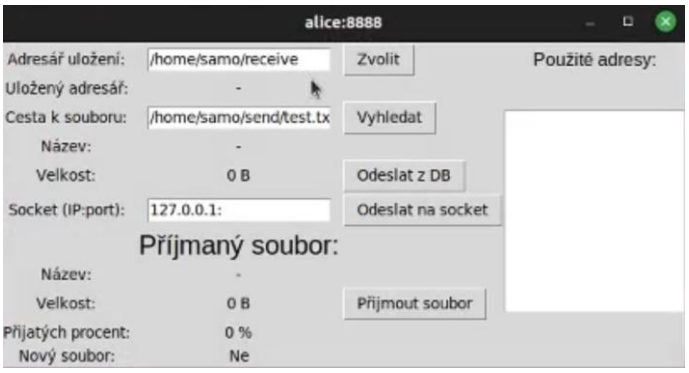
Grafické rozhraní je uděláno za pomoci knihovny tkinter. Po spuštění aplikace se objeví okno s názvem p2p app, do kterého musí uživatel vyplnit své jméno, heslo, port a čas uložení souboru. Výběr musí odsouhlasit kliknutím na tlačítko *ok*.




*Obrázek 10: vstupní grafické rozhraní*

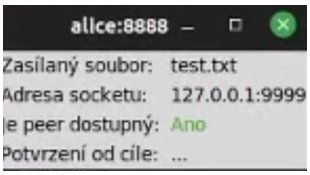
Následně se uživateli objeví jeho okno s názvem jeho zvoleného jména a čísla portu. Okno obsahuje pole na adresář, kde se má příchozí soubor uložit, potvrzení musí uživatel odkliknout tlačítkem *zvolit*. Pole na cestu k souboru, který chceme poslat, opět musí být potvrzeno kliknutím na tlačítko *vyhledat*. V řádcích pod cestou k souboru se objeví jméno souboru společně s jeho velikostí.

Následující pole slouží pro zadání IP adresy příjemce, pro odeslání musíme potvrdit kliknutím na *odeslat na socket*. Poslední pole slouží k přijímání souboru, kde uživatel musí souhlasit s přijímáním souboru a až poté dojde k přenosu. Na pravé straně okna se uživateli zobrazí, s kterými adresami již komunikoval.




*Obrázek 11: grafické rozhraní pro aplikaci na posílání a přijímání souborů*

Pokud uživatel pošle soubor kliknutím na *odeslat*, zobrazí se uživateli nové okno, které mu ukazuje úspěšnost jeho pokusu o poslání souboru a jestli je protější peer dostupný.




*Obrázek 12: grafické rozhraní pro kontrolu poslání souboru*

Obecné shrnutí grafického rozhraní v angličtině: source.gui package.

Přesné znění kódu s popisem v angličtině: MPC-KRY/source/gui/.

# POPIS SPUŠTĚNÍ APLIKACE

Program je doporučený pro verzi Python 3.8 a vyšší. Je doporučeno program spouštět v linuxovém OS případně v linuxovém virtuálním prostředí. Následně uživatel nainstaluje tkinter knihovnu (pro Debian/Ubuntu je to příkaz `APT-GET INSTALL PYTHON3-TK`). Následně uživatel nainstaluje potřebné python funkce ve složce obsahující program příkazem: `PIP3 INSTALL -R REQUIREMENTS.PY`. Spuštění aplikace následně bude fungovat skrz příkaz `./APP.PY`.

Návod na spuštění v angličtině najdete na github.com.

# ZÁVĚR

V semestrálním projektu byly splněny všechny části zadání. Byl zajištěn zabezpečený přenos souborů po síti mezi dvěma uživateli postaven na komunikaci P2P. Soubory se mohou posílat v reálném čase i se zpožděním, během něhož je přenášený soubor bezpečně uložen a čeká na odeslání. Výstupem projektu je aplikace s grafickým rozhraním. Klient má možnost si zvolit své jméno, heslo, port a čas, jak dlouho soubor bude čekat na odeslání před jeho zahozením. Ověření identity probíhá skrz certifikáty, které si strany vyměňují.