Documentation
Sensitive file sharing application

Martin Bezecný, Karel Ramiš, Marek Szymutko, Adam Tuček

Contents

1 Introduction

The aim of this project created for a university course named Cryptography was to create an application for sensitive file sharing. It is designed to be used by users who are on the same network and want to share encrypted files without the use of a server, that can be solved using peer-to-peer communication. The application is written in Python 3.12 with the use of supporting libraries, which will be described later in this introduction. The application is accompanied by a graphical user interface. It is designed to be used on any operating system that supports Python 3.12 and the required libraries.

The required files to run this application are all in this repository. Apart from the `main.py` file all of the code is in the file_share directory. The `main.py` file is the entry point of the application and is used to start the GUI. As input for the app there is nothing required apart from a file that a user wants to send. The output is a file that was sent to the user from another user using the app.

The application uses the following libraries:

- `Tkinter (v8.6)` - for the Graphical User Interface
- `Uvicorn (v0.27.1)` - used for receiver realisation
- `FastAPI (v0.110.0)` - for the receiver API
- `SQLAlchemy (v2.0.28)` - for the database implementation
- `Cryptography (v42.0.5)` - for encryption and decryption of files
- `AIOHTTP (v3.9.3)` - for communication between the sender and the receiver

2 Installation

Installation is done using the following steps for RPM based system. For APT based systems, replace `dnf` with `apt`. For MacOS use Homebrew package manager.

```
sudo dnf install python3 python3-pip python3-virtualenv
python3 -m venv .venv
source .venv/bin/activate
python3 -m pip install pdm
pdm install
```

To launch the application itself, you can use the following command:

```
python3 main.py
```

3 GUI

The GUI starts by running command `python3 main.py` in the virtual environment.

3.1 Login screen

When the app is opened, the user is greeted by a login screen. The user can either log in with an existing account if they used the app in the past as shown in Figure 1 or create a new one with username and password as in Figure 2.

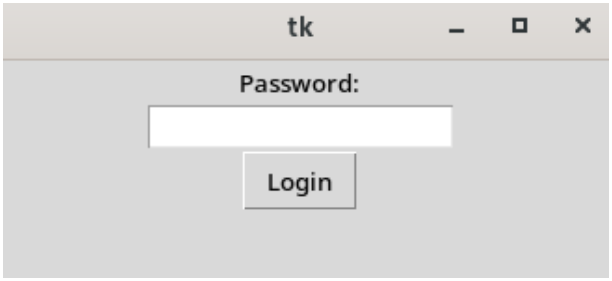

Figure 1: Login screen

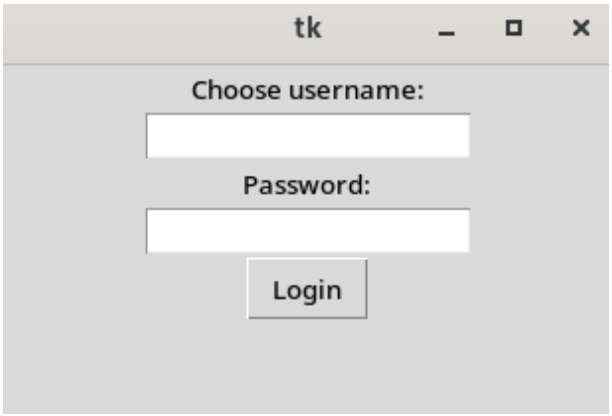

Figure 2: Register screen

3.2 Main window

After the login, the user will see the main application window.

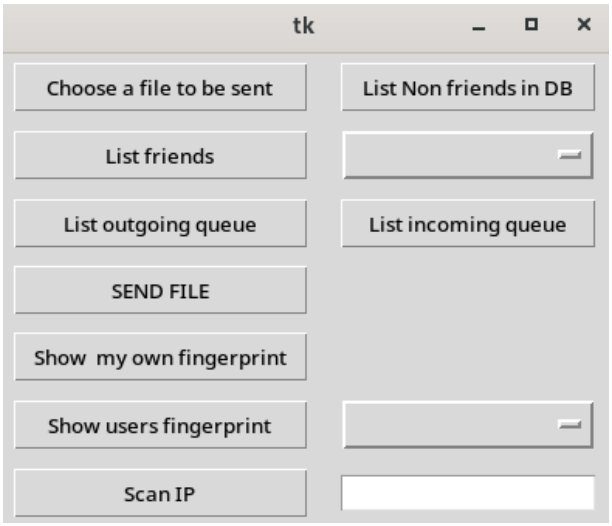

Figure 3: Main window

3.2.1 File selection

Selecting a file for transfer is done by using the button `Choose a file to be sent` that is shown in Figure 3. After pressing the button, the file selection window will open and user can select the file they want to send.

3.2.2 User discovery and friend addition

There are 2 ways to discover other users who are using the app. Either the user can scan the whole network for active apps that are running using `List Non friends in DB` button.

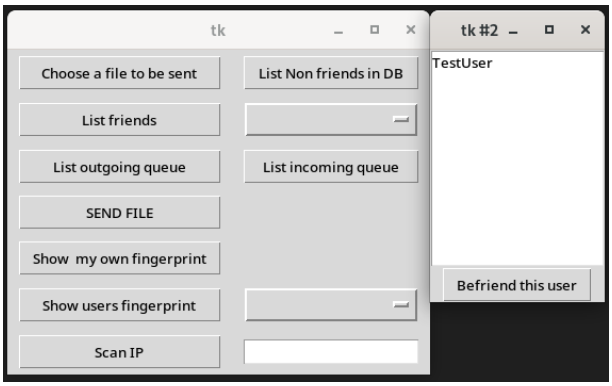

Figure 4: Adding users to db by scanning the network

The other way is by using button `Scan IP`. User enters the address they want to scan in the text field next to the button and then presses the button.

Figure 5: Adding user to db by scanning given IP

Adding friends is handled by pressing the button `Befriend this user` that we can see in Figure 4.

3.2.3 Listing of users

There are 2 ways to list other users that the user can interact with. Either list only those users who the user added to their friend list or list all users in database. For this there are 2 buttons shown on Figure 3. Those are `List friends` and `List Non friends in DB` respectively.

3.2.4 Listing queued files

Files that cannot be sent immediately are queued and will be sent once the user is available. The user can see the list of queued files by pressing the button `List outgoing queue`.

3.2.5 Friend selection

Choosing a friend who will be the target of the file transfer is done by choosing one friend from the drop-down menu next to the `List friends` button.

3.2.6 Fingerprints

There are 2 buttons regarding the fingerprints. The first one is `Show my own fingerprint` so the user can look up their certificate fingerprint. The `Show users fingerprint` button is paired with a drop-down menu, where the user can choose whose fingerprint they want to see in case they want to verify that the fingerprint is correct and they are not falling victim for some kind of MitM attack.

3.2.7 File sharing

After selecting the file and the friend to send it to, the user can press the button `Send file` to send the file to the selected friend. If the target is not available, the app will prompt the user with following message and put the file in the queue to be sent later.

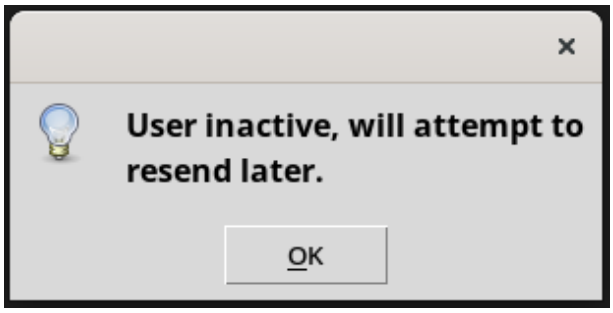

Figure 6: Target not available

Receiving the file is done by the target user pressing the button `List incoming queue` which opens window with all files that are waiting to be received. The user can either save or ignore a single file or accept all files in the queue.

Figure 7: Incoming queue

If the user chooses to save the file they are prompted with a window to select the location where the file will be saved.

4 Functionality description

This section goes over the app functionalities and how they are designed and implemented.

4.1 Login

This is a section going over logging in to the application. If it is the first time the user uses the application they will have to create an account with username and password. Otherwise only password is required. If user inserts a password that does not match their corresponding password stored in database the "Invalid password" message is displayed. If the password is correct, the main window of the application is shown.

This is implemented in the file `main.py` in the function `start_app()`.

4.2 File selection

Selecting files is handled by the button `Choose a file to be sent`. When the button is pressed, a dialog window is opened where the user can select a file. The implementation of this is done using module `filedialog` from `tkinter` library. If a file is not selected, the window won't let them select a path that is not pointing to a file. If the window is closed without selecting a file, the filepath is kept empty.

The code itself can be seen in file `file_share/app/app.py` in the function `get_file()`.

4.3 Listing users

There are two functionalities that are related to listing users. The first one is listing all non-friend users that are in the database. The second one is listing all users that the user has added as friends. Both of these open new windows where the users are listed. The difference between these two windows is that in the window where non-friend users are listed, there is a button to add them as a friend.

The list friends functionality is simpler and it is done by retrieving all users from the database and filtering out the ones that are friends with the user. The method that handles this is the `get_all_users` from `file_share/database/__init.py__`, where there is friend parameter which is by default set to `True`.
The non-friends is a bit more complicated. First, the list of all users except friends is collected from the database. It is listed in the window. If user wants to add a friend, they select that user from the list and press button "Befriend this user", this calls a function `befriend` in `file_share/database/__init.py__`. This adds the user as a friend by setting the `is_friend` attribute to `True`.

4.4 File sending initialization

For this core functionality there is a flow chart attached for easier comprehensibility. It is implemented in function `send_file` in `file_share/app/app.py`. The function first checks if the file for transfer is selected, if that is true, the file is then prepared for transfer and file transfer is attempted. Sending is done by function `send_or_store_file` from `file_share/sender/sender.py`. As a return there is a `SendStatus` object, which is enum created for this application. There are 5 states that the object can be in, those can be viewed in `file_share/definitions/enums.py`. Based on the state of the user which is passed to the function, the file will either be sent, stored in outgoing queue or nothing will happen.

If the file is to be stored then it is scheduled to be added to files table in the database using `store_file` from `file_share/database/__init.py__`.

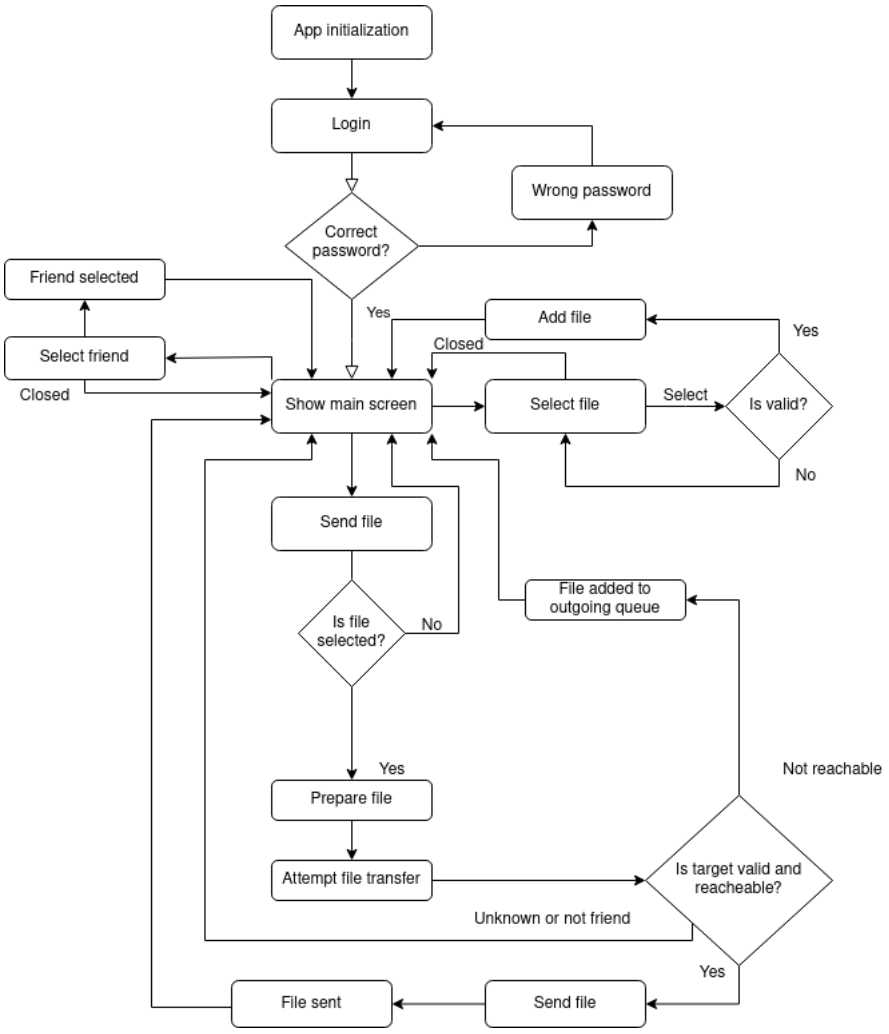

Figure 8: Flowchart of file sending initialization

4.5 File sending

In this section the process of file transfer itself is described. It is accompanied by a figure that shows the sides of the communication and the steps that each of the sides takes.
This whole process is based on using a one time use API key which is generated by the receiving side of the communication. This key is then used by the sender in the header of the file transfer as a verification of the communication. Firstly the sender extracts from the file object that is passed as a parameter the username of target user, then on the basis of this username retrieves the address of this user from database, unless the file has address override in it. In which case the override address is used as a target address.
Then the SSL context of given username is taken and using aiohttp session a request is made to the target user. The target then verifies if the sender is a friend, if not then exception 401 is raised and the communication is terminated by this exception. If the sender is a known friend, then an API key is generated, encoded using the senders public key extracted from their certificate and sent back to the sender as a response. This verification and key generating is done in `file_share/receiver/receiver_api.py` in method `auth()`.
When sender receives this API key, it is decrypted using the senders private key, then a FormData object is created, the file is added to it and finally a sending is done with this FormData object, where the API key is included as the header.
On the receiving side, when the file transfer is done the API key is removed from database as it is one time use only. If the key is not found in database then a 401 exception is raised and this ends the communication. Otherwise a new file object is created with the incoming attribute set to True, the object is timestamped, and sender username is associated with it. Then it is added to the database table and the file's name is returned as a response.

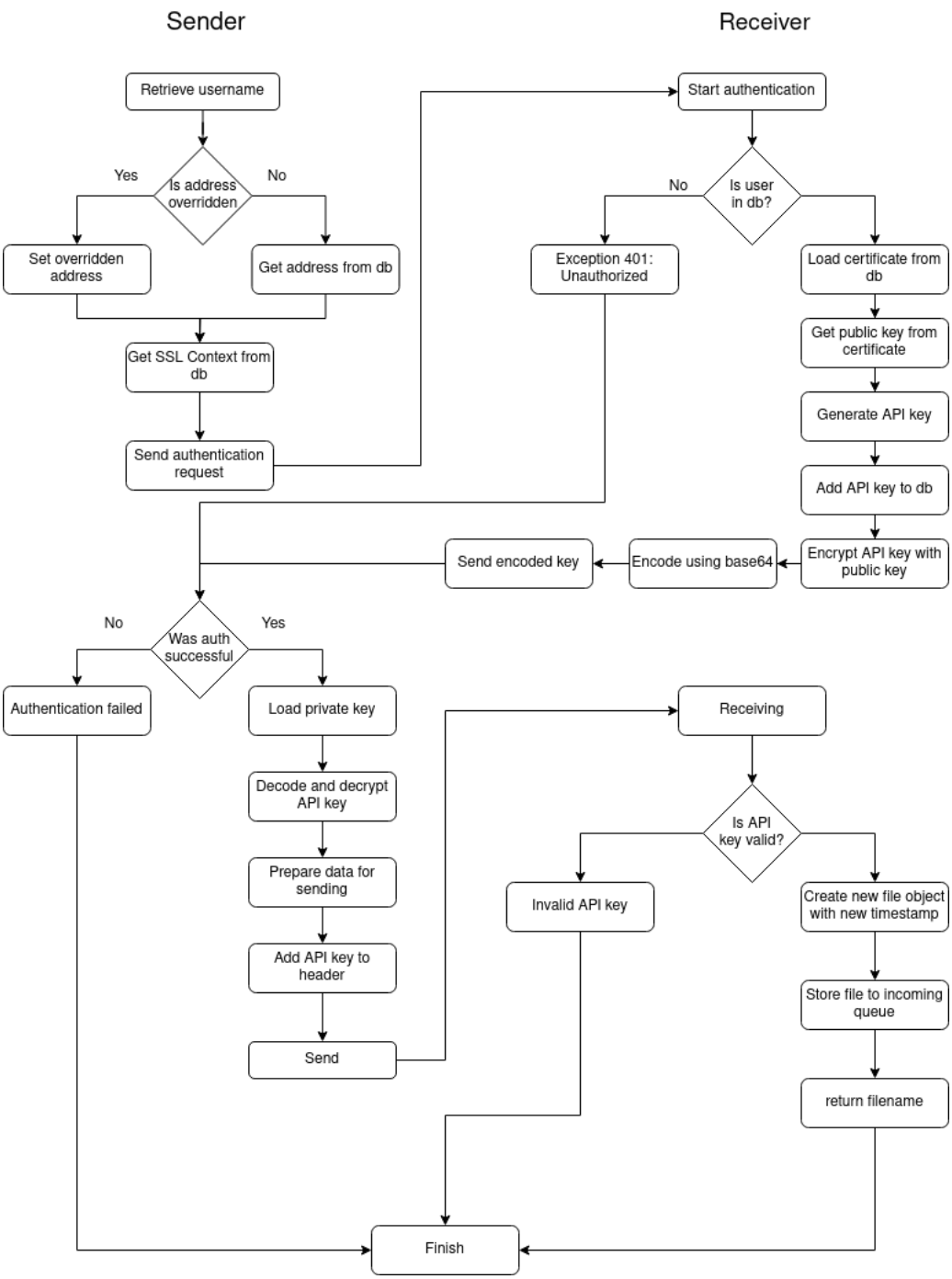

Figure 9: Flowchart of file transfer

5 Application structure

```
KRY-Project
├── file_share
│   ├── app ..... Core methods of the application
│   │   ├── __init__.py
│   │   ├── app.py
│   │   ├── init_app.py
│   │   └── test_send_file.py
│   ├── database ..... Definitions of all database tables
│   │   ├── __init__.py
│   │   ├── base.py
│   │   ├── files.py
│   │   ├── keys.py
│   │   ├── me.py
│   │   └── users.py
│   ├── definitions
│   │   ├── __init__.py
│   │   ├── dataclasses.py
│   │   ├── enum.py
│   │   └── procedures.py
│   ├── friend_finder ..... Methods to discover and work with users on the network
│   │   ├── __init__.py
│   │   └── ping_em.py
│   ├── receiver ..... Endpoint definitons enabling data reception with authentication
│   │   ├── __init__.py
│   │   ├── api_keys.py
│   │   ├── get_ip.py
│   │   └── receiver_api.py
│   ├── sender ..... Methods to send files and work with queue
│   │   ├── __init__.py
│   │   ├── sender.py
│   │   └── ssl_context.py
│   └── __init__.py
├── main.py
├── pdm.lock ..... Package data
├── pyproject.toml ..... Basic configuration file
├── README.md
└── testfile.txt
```

6 Conclusion

This project set out to create a GUI controlled application that enables a secure Peer-to-Peer file transfer on a public network in two modes:

– Real-time transmission – both users are online and the transmission is immediate
– Delayed transmission – the receiving user is offline so the transmission is stored in a secured storage waiting to be sent once the user becomes available

All of the aforementioned goals were achieved using the Python programming language.