

Applied Cryptography Project Seminar Project Documentation

Korpa Bence JDJKCH

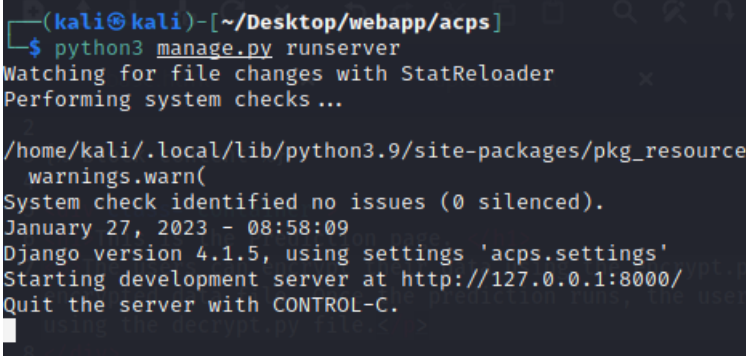
1. The Goal of the Project

The goal was to create a website that provides an interactive interface for a secure Machine Learning Prediction. To achieve this, I have used Django 4.1.5

(<https://www.djangoproject.com/>) for creating the website, bootstrap for the very basic design, and for the secure Machine Learning part I have used the Concrete ML framework (<https://docs.zama.ai/concrete-ml/>) that is based on Python.

2. Running the server

The server can be run using the `python3 manage.py runserver` command. To run the server the used tools (Concrete-ML and Django) should be pre-installed.

A terminal window with a dark background and light-colored text. The prompt is `(kali㉿kali)-[~/Desktop/webapp/acps]`. The user enters `$ python3 manage.py runserver`. The output shows: `Watching for file changes with StatReloader`, `Performing system checks ...`, a warning from `/home/kali/.local/lib/python3.9/site-packages/pkg_resource` about a system check, `System check identified no issues (0 silenced).`, the date `January 27, 2023 - 08:58:09`, `Django version 4.1.5, using settings 'acps.settings'`, `Starting development server at http://127.0.0.1:8000/`, and `Quit the server with CONTROL-C.`

```
(kali㉿kali)-[~/Desktop/webapp/acps]
$ python3 manage.py runserver
Watching for file changes with StatReloader
Performing system checks ...

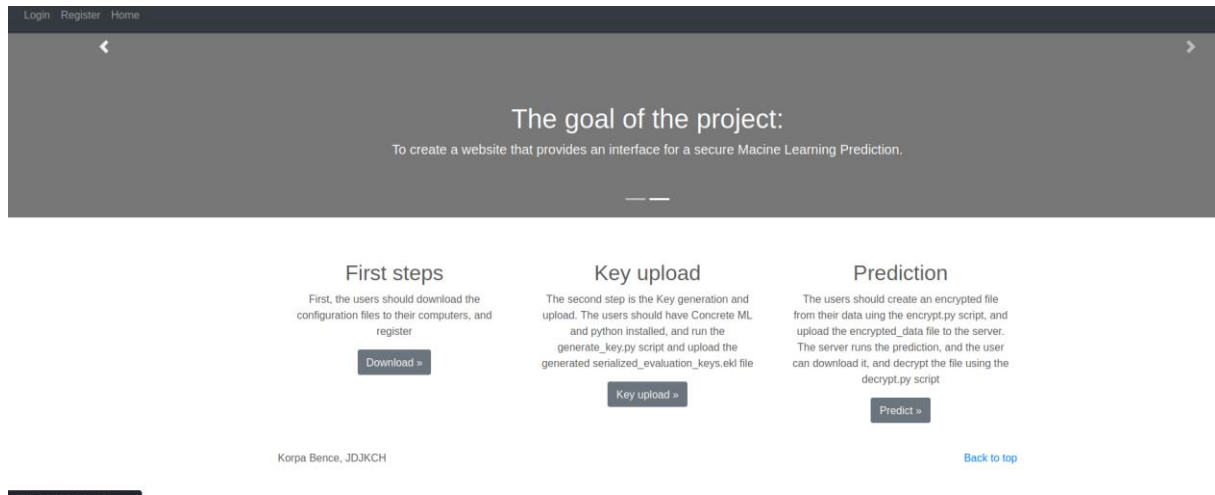
/home/kali/.local/lib/python3.9/site-packages/pkg_resource
warnings.warn(
System check identified no issues (0 silenced).
January 27, 2023 - 08:58:09
Django version 4.1.5, using settings 'acps.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.
```

Picture 1. starting the webserver

I did not deploy the server onto an actual server, only used the localhost setting for testing, but it is usually straightforward with Django.

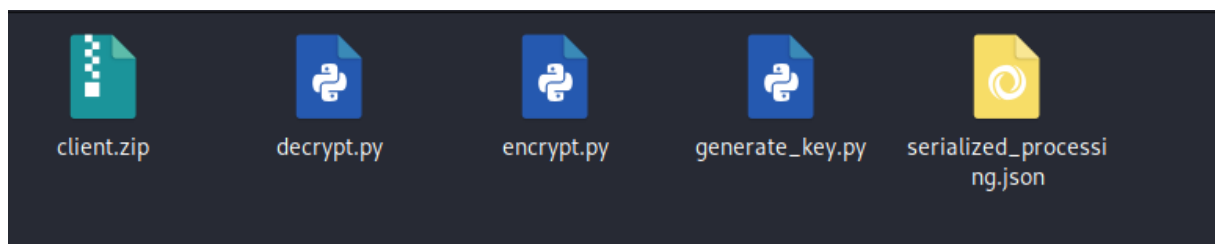
3. User Documentation

After the webserver is running it can be reached, and a homepage shows:



Picture 2: Homepage

From the homepage the instructions to use the website are clear, first the user should download the configuration files, and scripts. It is a zip file that after extraction contain the following files:



Picture 3: Downloaded configuration files, and scripts.

The client.zip file should not be further extracted. Then the user should register for the site, by clicking on the register link

Sign up

Username: Required. 150 characters or fewer. Letters, digits and @/./+/-/_ only.

Password:

- Your password can't be too similar to your other personal information.
- Your password must contain at least 8 characters.
- Your password can't be a commonly used password.
- Your password can't be entirely numeric.

Password confirmation: Enter the same password as before, for verification.

[Sign Up](#)

Korpa Bence, JDJKCH

Picture 4: Registration

After a successful registration, the website redirects to the Login page, that is available from the homepage as well:

Login Register Home

Log In

Username:

Password:

[Log In](#)

Korpa Bence, JDJKCH

Picture 5: Login

After Logging in the homepage shows again, but with different links. Logging in is the only way to reach the key upload, and prediction pages.

Logout Home Key Upload Predict

The goal of the project:

To create a website that provides an interface for a secure Machine Learning Prediction.

First steps

First, the users should download the configuration files to their computers, and register

[Download »](#)

Key upload

The second step is the Key generation and upload. The users should have Concrete ML and python installed, and run the generate_key.py script and upload the generated serialized_evaluation_keys.ekd file

[Key upload »](#)

Prediction

The users should create an encrypted file from their data using the encrypt.py script, and upload the encrypted_data file to the server. The server runs the prediction, and the user can download it, and decrypt the file using the decrypt.py script

[Predict »](#)

Korpa Bence, JDJKCH [Back to top](#)

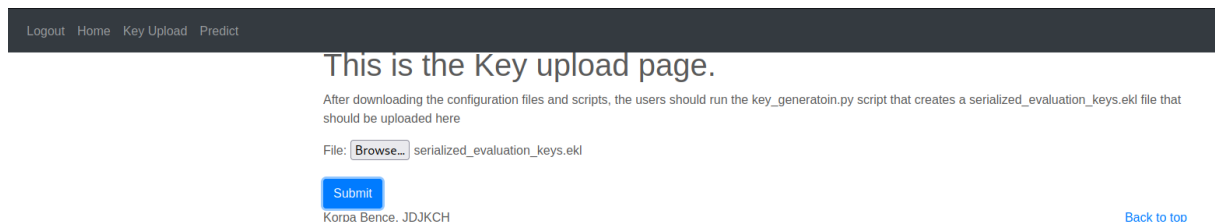
Picture 6: Homepage after Login

The Next step for the user is the key generation. From the extracted file earlier the generate_key.py should be run, it takes about 15-30 seconds:

```
(kali@kali)-[~/TEST/client_data(3)]
$ python3 generate_key.py
/home/kali/.local/lib/python3.9/site-packages/pkg_resources/
warnings.warn(
KeySetCache: miss, regenerating ./3285928842239850127/0_0
(kali@kali)-[~/TEST/client_data(3)]
$
```

Picture 7: Key generation

After running the script, a new folder appears that contains the private keys, and a new file, serialized_evaluation_keys.ekl is created. This file should be uploaded to the key upload page:

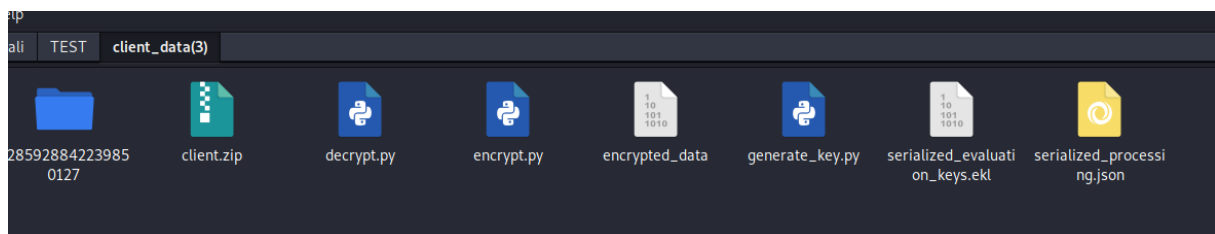


Picture 8: Key upload page

After the upload the user should encrypt the data using the encrypt.py script, that creates an encrypted_data file:

```
(kali@kali)-[~/TEST/client_data(3)]
$ python3 encrypt.py
/home/kali/.local/lib/python3.9/site-packages/p
warnings.warn(
```

Picture 9: Running the encrypt.py



Picture 10: User's folder after encrypting the data.

This encrypted data should then be uploaded to the Prediction site, then the server performs the prediction and the user downloads the encrypted prediction:

This is the Prediction page.

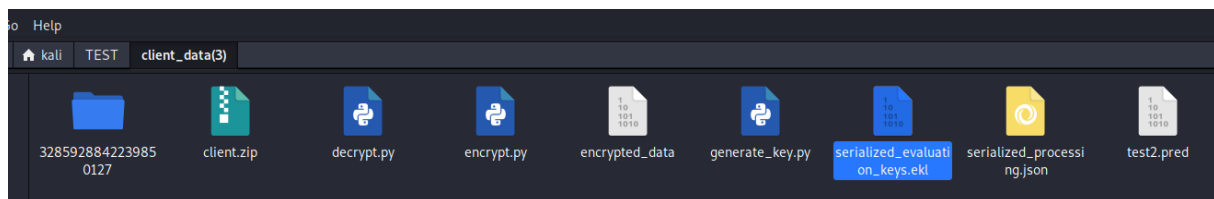
The users can encrypt their data using the encrypt.py script, and upload the generated encrypted_data file. Once the prediction runs, the users download the result that can be decrypted using the decrypt.py file.

File: encrypted_data

Korpa Bence, JDJKCH

[Back to top](#)

Picture 11: Prediction Page



Picture 12: User's folder after downloading the predicted result

As it can be seen, test2.pred was downloaded from the server. Now the user should run the decrypt.py script and as first argument the name of the encrypted prediction should be given, and the prediction will be decrypted:

```
(kali@kali)-[~/TEST/client_data(3)]
$ python3 decrypt.py test2.pred
/home/kali/.local/lib/python3.9/site-packages/pkg_resources/__init__.py:
warnings.warn(
[0.12789139 0.87210861]
```

Picture 13: Decoded Prediction.

The meaning of the decoded prediction, and the exact ML task that was used will be explained in the next section.

The users can generate new keys any time they want, running the generate_key.py file, and upload it again, but it is not necessary, and as the key files are quite big (100MB) it is probably not that important. If the users have new data, they can open the encrypt.py script and exchange the clear_input variable to their instance that they would like a prediction on. This is also explained in more details in the next section. After that the encryption, upload, prediction, decryption steps can be done multiple times.

4. Machine Learning task

The Machine Learning's source code is found on the server in the following python scripts:

- For the training training.py in the acps/server folder
- For the saving of the keys in the acps/save_key.py file
- For the prediction in the acps/predict_data.py file

Training the Model

As it can be seen from the training.py file, I have used the preloaded breast_cancer dataset from the sklearn library. This dataset contains 569 instances with 10 features each, and 2 classes to decide (Malignant or Benign). For the training XGBClassifier was used, but other ML algorithms could be used as well. Running this training.py file creates the server.zip file that has the trained server that is used in the backend, and creates the client configuration files that are put into the archive that is downloaded by the User. This training.py file should not be uploaded as a source code normally, only the server.zip after training the data locally to ensure that no information is learned about the model.

The model can be loaded when the prediction happens. At this point the Concrete ML library is used that makes fully homomorphic machine learning possible.