Configuration Manual

MSc Research Project
Masters in Cybersecurity

Victoria Mfon Atauba
Student ID: x20122837

School of Computing
National College of Ireland

Supervisor: Dr Niall Heffernan

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name: Victoria Atauba Mfon

Student ID: x20122837

Programme: Cybersecurity **Year:** 2022

Module: MSc Research Project

Lecturer: Dr Niall Heffernan

Submission Due Date: 15/08/2022

Project Title: A study on preserving privacy of Internet of Vehicles (IOV) data in cloud infrastructure using homomorphic encryption

Word Count: **Page Count:**

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: Victoria Atauba Mfon

Date: 15/08/2022

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

CONFIGURATION MANUAL: A STUDY ON PRESERVING PRIVACY OF INTERNET OF VEHICLE (IOV) DATA IN CLOUD INFRASTRUCTURE USING HOMOMORPHIC ENCRYPTION

1 INTRODUCTION

The configuration manual provides a thorough explanation of the step-by-step procedure used to execute the solution proposed in this study. The setup of the simulation environment, methods, software tools, applications, and hardware features used throughout the implementation process are also covered in this manual.

2 HARDWARE FEATURES

Device name	HP pavilion power notebook
OS type	64-bit processor
OS version	Windows 10
Processor	Intel core
RAM	16gb
Hard drive	1 terabyte

2.1 SOFTWARE TOOLS

The proposed solution was implemented using python programming language for both the server and client side and was run on Anaconda platform. One significant library used for this implementation is the pyfhel library which is incorporated in the programming language.

2.3 SOFTWARE INSTALLATION

The step-by-step process for installing the software used include:

i Download anaconda from: https://www.anaconda.com/

ii After download, open anaconda and install dependencies.

iii To install dependencies, create new anaconda environment having python 3.8 version. The following is the process for the creation of the environment: conda create --environment_name python 3.8.

iv Activate the conda environment using: conda activate environment_name.

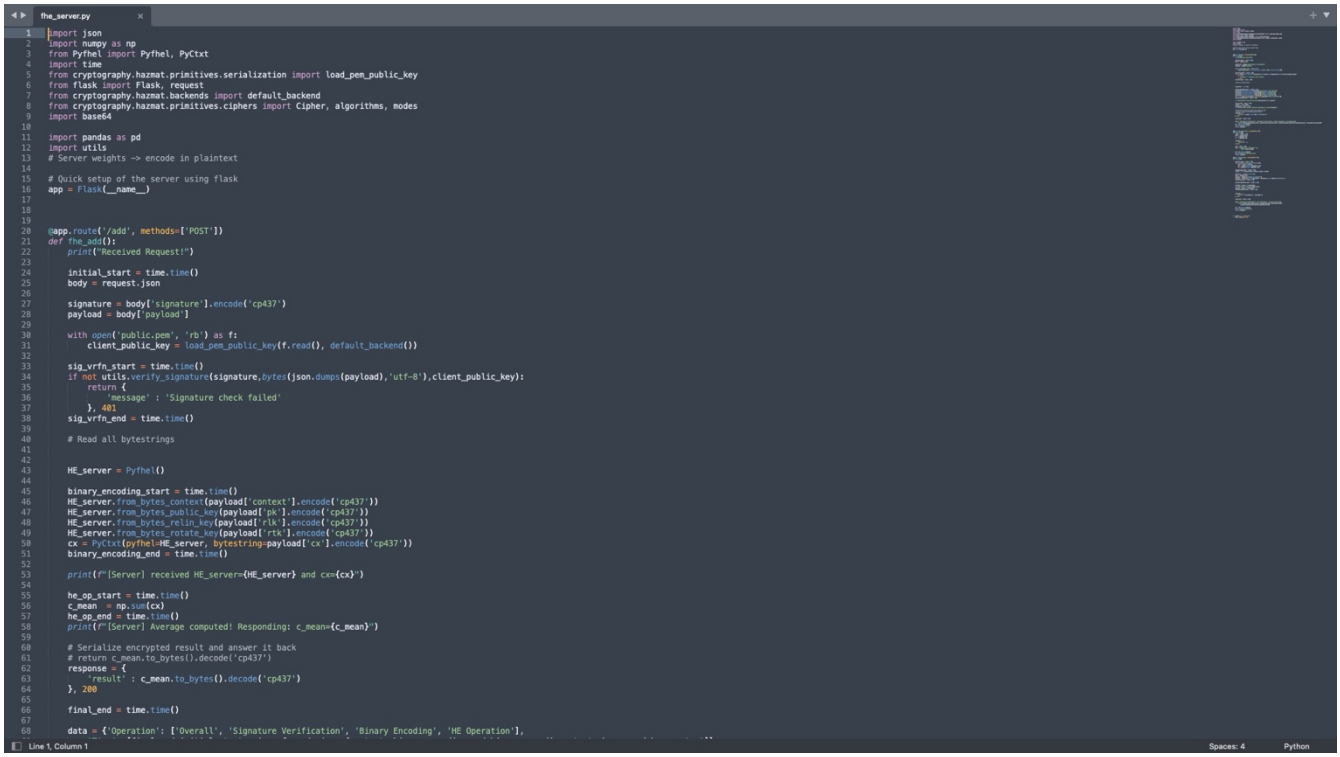
v Install pip dependencies using: pip install -r requirements.txt

vi Launch the homomorphic encryption processing server using: python fhe_server.py

vii Run the vehicle client program using: python client.py

3 DESCRIPTIONS

This application models a simulation between a server running locally on the machine and vehicle client sending data for processing. The server has separate HTTP routes for each operation it supports. For example, to compute the dot product of two vectors, it would have a route of '/dot'.



```python
import json
import numpy as np
from Pyfhel import Pyfhel, PyCtxt
import time
from cryptography.hazmat.primitives.serialization import load_pem_public_key
from flask import Flask, request
from cryptography.hazmat.backends import default_backend
from cryptography.hazmat.primitives.ciphers import Cipher, algorithms, modes
import base64

import pandas as pd
import utils
# Server weights -> encode in plaintext

# Quick setup of the server using flask
app = Flask(__name__)


@app.route('/add', methods=['POST'])
def fhe_add():
    print("Received Request!")

    initial_start = time.time()
    body = request.json

    signature = body['signature'].encode('cp437')
    payload = body['payload']

    with open('public.pem', 'rb') as f:
        client_public_key = load_pem_public_key(f.read(), default_backend())

    sig_vrfn_start = time.time()
    if not utils.verify_signature(signature,bytes(json.dumps(payload),'utf-8'),client_public_key):
        return {
            'message' : 'Signature check failed'
        }, 401
    sig_vrfn_end = time.time()

    # Read all bytestrings


    HE_server = Pyfhel()

    binary_encoding_start = time.time()
    HE_server.from_bytes_context(payload['context'].encode('cp437'))
    HE_server.from_bytes_public_key(payload['pk'].encode('cp437'))
    HE_server.from_bytes_relin_key(payload['rlk'].encode('cp437'))
    HE_server.from_bytes_rotate_key(payload['rtk'].encode('cp437'))
    cx = PyCtxt(pyfhel=HE_server, bytestring=payload['cx'].encode('cp437'))
    binary_encoding_end = time.time()

    print(f"[Server] received HE_server={HE_server} and cx={cx}")

    he_op_start = time.time()
    c_mean  = np.sum(cx)
    he_op_end = time.time()
    print(f"[Server] Average computed! Responding: c_mean={c_mean}")

    # Serialize encrypted result and answer it back
    # return c_mean.to_bytes().decode('cp437')
    response = {
        'result' : c_mean.to_bytes().decode('cp437')
    }, 200

    final_end = time.time()

    data = {'Operation': ['Overall', 'Signature Verification', 'Binary Encoding', 'HE Operation'],
```

Fig 1: Snapshot of the server side

The vehicle client homomorphically encrypts the data and depending upon the operation it wants the server to perform, it sends the payload to the corresponding route. The server performs the requested operation without encrypting the incoming payload and returns the result back to the client.

client.py

```python
        adversay_private_key = utils.create_private_key()
        r = requests.post('http://127.0.0.1:5000/add',
        json={
            'payload' : 'modified_paylaod' if MODIFY_PAYLOAD else payload,
            'signature': utils.sign_message(bytes(json.dumps(payload),'utf-8'),adversay_private_key).decode('cp437')
        })
    else:
        r = requests.post('http://127.0.0.1:5000/add',
            json={
                'payload' : 'modified_paylaod' if MODIFY_PAYLOAD else payload,
                'signature': utils.sign_message(bytes(json.dumps(payload), 'utf-8'),private_key).decode('cp437')
            })


    if r.status_code == 401:
        print(f'Signature check failed')
        exit(-1)

    c_res = PyCtxt(pyfhel=HE_client, bytestring=r.json()['result'].encode('cp437'))

    res = HE_client.decryptFrac(c_res)


    # The expected result and result from FHE can be different because of noise
    print(f"[Client] Response received! Result is {np.round(res[3], 4)}")

def basic_add():
    paylaod =  {
        'x': 10,
        'y': 20
    }
    r = requests.post('http://127.0.0.1:5000/basic-add',
        json=paylaod)

    print(f'Response: {r.json()}')

def aes_add():
    payload = json.dumps({
        'x': 10,
        'y': 20
    })

    with open('aes-keys.json','r') as file:
        data = json.load(file)
        key = base64.b64decode(data['key'])
        iv = base64.b64decode(data['iv'])

    cipher = Cipher(algorithms.AES(key), modes.CBC(iv))
    encryptor = cipher.encryptor()
    message = utils.pad(payload.encode('ascii'))
    ct = encryptor.update(message) + encryptor.finalize()


    r = requests.post('http://127.0.0.1:5000/aes-add',
        json={
        'payload': ct.decode('cp437')
    })

    print(f'Response: {r.json()}')


fhe_add()
# aes_add()
# basic_add()
```

Line 1, Column 1 Spaces: 4 Python

Fig 2: Snapshot of the client side