National College of Ireland

Configuration Manual

MSc Research Project
Programme Name

Mamidi Snehith Reddy
Student ID: 22195297

School of Computing
National College of Ireland

Supervisor: Jawad Salahuddin

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name: MAMIDI SNEHITH REDDY

Student ID: 22195297

Programme: MSc CYBER SECURITY **Year:** 2023-2024

Module: MSc Research Practicum Part 2

Lecturer: Jawad Salahuddin

Submission Due Date: 12 August 2024

Project Title: Ensuring Privacy in the Digital Era: Innovative Strategies for Data Encryption (ISDE) with Access Control

Word Count: 409 **Page Count:** 6

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: Mamidi Snehith Reddy

Date: 12/8/2024

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

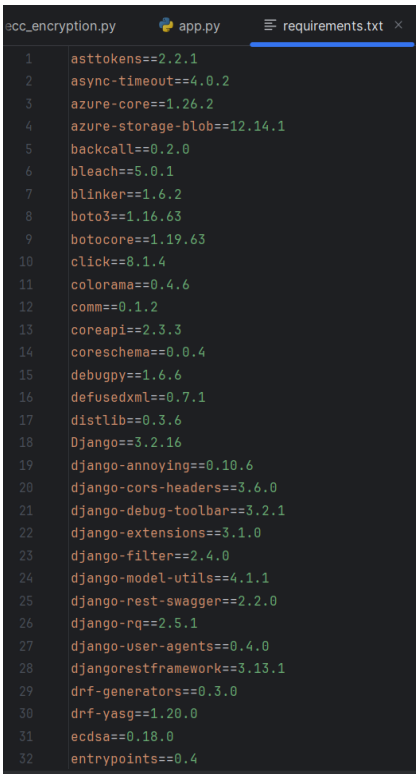
Mamidi Snehith Reddy
Student ID: 22195297

1 Introduction:

- Welcome to the configuration manual for the data encryption with access control. It is a python-based implementation which encrypts solutions often impose high computational overhead, making them less practical for widespread use and inadequate for addressing evolving security threats in cloud environments. This manual is divided into several sections of the project which includes system configuration, code setup. This will help to deploy and manage the project.

2 System Configurations:

Python: Version 3.12
Libraries: Install all the libraires using the command below pip install -r requriemnts.txt



Anaconda setup:

Download Anaconda from the online website link given below:

- https://www.anaconda.com/download

- Install the .exe and run the program.

3 Code Setup

app.py

```
from flask import Flask, render_template, request, redirect, url_for, session, flash
from datetime import datetime
from flask_mysqldb import MySQL
from flask_mail import Mail, Message
from werkzeug.utils import secure_filename
from lib_file import lib_path
from rsa_encryption import get_keys, rsa_encryption, rsa_decryption
from ecc_encryption import generate_keys, ecc_encryption, ecc_decryption
import os
import ftplib
import nltk
import mysql.connector
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from collections import Counter

application = Flask(__name__)
application.secret_key = 'new'

application.config['MYSQL_HOST'] = 'localhost'
application.config['MYSQL_USER'] = 'root'
application.config['MYSQL_PASSWORD'] ='0812'
application.config['MYSQL_DB'] = 'ehr_tf_final'
```

ECC Decryption Function Code:

```python
import hashlib
import secrets
from Crypto.Cipher import AES
from tinyec import registry
import pickle as pkl
from hashlib import md5
from os import urandom
import random

curve = registry.get_curve('brainpoolP256r1')


3 usages
def generate_keys():
    private_Key = secrets.randbelow(curve.field.n)
    public_key = private_Key * curve.g
    return private_Key, public_key


2 usages
def ecc_point_to_256_bit_key(point):
    sha = hashlib.sha256(int.to_bytes(point.x, 32, 'big'))
    sha.update(int.to_bytes(point.y, 32, 'big'))
    return sha.digest()


1 usage
def encrypt_AES_GCM(msg, secretKey):
    aesCipher = AES.new(secretKey, AES.MODE_GCM)
    ciphertext, authTag = aesCipher.encrypt_and_digest(msg)
    return ciphertext, aesCipher.nonce, authTag
def encrypt_ECC(msg, pubKey):
    msg = bytes(msg, 'utf-8')
    pubKey = int(pubKey) * curve.g
    ciphertextPrivKey = secrets.randbelow(curve.field.n)
    sharedECCKey = ciphertextPrivKey * pubKey
    secretKey = ecc_point_to_256_bit_key(sharedECCKey)
    ciphertext, nonce, authTag = encrypt_AES_GCM(msg, secretKey)
    ciphertextPubKey = ciphertextPrivKey * curve.g
    return ciphertext, nonce, authTag, ciphertextPubKey


1 usage
def decrypt_AES_GCM(ciphertext, nonce, authTag, secretKey):
    aesCipher = AES.new(secretKey, AES.MODE_GCM, nonce)
    plaintext = aesCipher.decrypt_and_verify(ciphertext, authTag)
    return plaintext


1 usage
def decrypt_ECC(encrypted_msg, p2_key):
    (ciphertext, nonce, authTag, ciphertextPubKey) = encrypted_msg
    sharedECCKey = int(p2_key) * ciphertextPubKey
    secretKey = ecc_point_to_256_bit_key(sharedECCKey)
    plaintext = decrypt_AES_GCM(ciphertext, nonce, authTag, secretKey)
    return plaintext
```

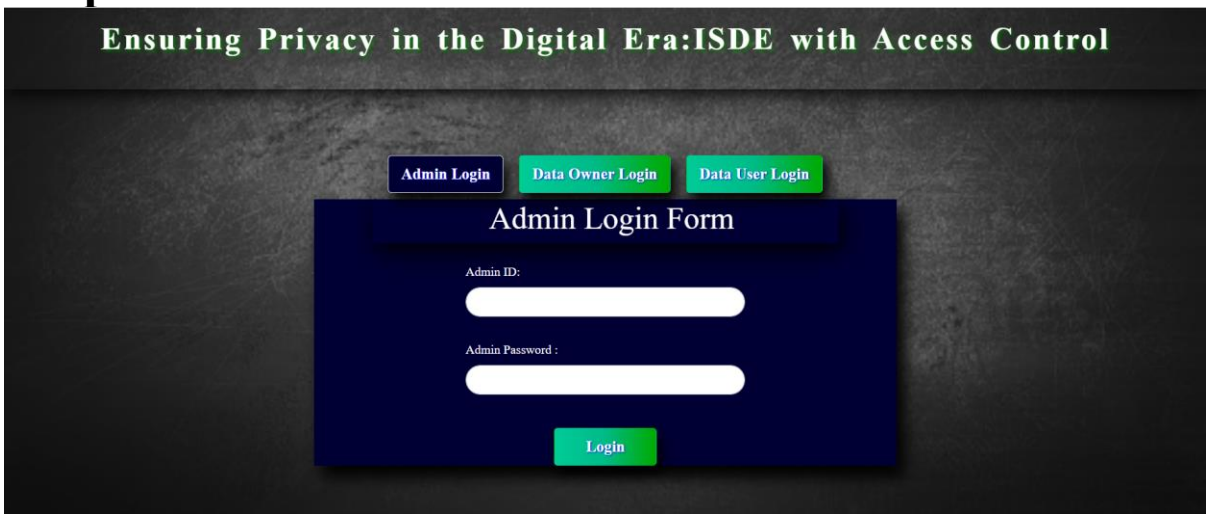
```python
def ecc_encryption(in_file, out_file, public_key, p_file_name, key_length=32):
    pickle_file_name = p_file_name.split('.')[0] + '.pkl'
    bs = AES.block_size  # 16 bytes
    salt = urandom(bs)  # return a string of random bytes
    password = str(random.randint(1000, 10000))
    key, iv = derive_key_and_iv(password, salt, key_length, bs)
    cipher = AES.new(key, AES.MODE_CBC, iv)
    out_file.write(salt)
    finished = False

    while not finished:
        chunk = in_file.read(1024 * bs)
        if len(chunk) == 0 or len(chunk) % bs != 0:  # final block/chunk is padded before encryption
            padding_length = (bs - len(chunk) % bs) or bs
            chunk += str.encode(padding_length * chr(padding_length))
            finished = True
        out_file.write(cipher.encrypt(chunk))
    encrypted = encrypt_ECC(password, public_key)
    file_path = "static/pickle_files/" + pickle_file_name
    with open(file=file_path, mode="wb") as file:
        pkl.dump(obj=encrypted, file=file)
    return file_path


2 usages
def ecc_decryption(in_file, out_file, pkl_filename, private_key, key_length=32):
    pickle_file_name = pkl_filename.split('.')[0] + '.pkl'
    pickle_file_location = 'static/pickle_files/' + pickle_file_name
    with open(file=pickle_file_location, mode="rb") as file:
        res_msg = pkl.load(file=file)

def ecc_decryption(in_file, out_file, pkl_filename, private_key, key_length=32):
    pickle_file_name = pkl_filename.split('.')[0] + '.pkl'
    pickle_file_location = 'static/pickle_files/' + pickle_file_name
    with open(file=pickle_file_location, mode="rb") as file:
        res_msg = pkl.load(file=file)

    dec_text = decrypt_ECC(res_msg, private_key)
    dec_text = dec_text.decode()
    bs = AES.block_size
    salt = in_file.read(bs)
    key, iv = derive_key_and_iv(str(dec_text), salt, key_length, bs)
    cipher = AES.new(key, AES.MODE_CBC, iv)
    next_chunk = ''
    finished = False
    while not finished:
        chunk, next_chunk = next_chunk, cipher.decrypt(in_file.read(1024 * bs))
        if len(next_chunk) == 0:
            padding_length = chunk[-1]
            chunk = chunk[:-padding_length]
            finished = True
        out_file.write(bytes(x for x in chunk))
```

RSA Encryption and Decryption Code:

rsa_encryption.py | ecc_encryption.py | app.py | requirements.txt | ehr_final_m_admin.sql | ehr_final_m_d

```python
import random
from hashlib import md5
from Crypto.Cipher import AES
from os import urandom
import rsa


# import docx
# import PyPDF2


3 usages
def get_keys():
    public_key, private_key = rsa.newkeys(128)
    print(public_key)
    print(private_key)
    pub_key = public_key.save_pkcs1()
    pri_key = private_key.save_pkcs1()
    pub_key_str = pub_key.decode()
    pri_key_str = pri_key.decode()
    return pub_key_str, pri_key_str


2 usages
def derive_key_and_iv(password, salt, key_length, iv_length):  # derive key and IV from password and salt.
    d = d_i = b''
    while len(d) < key_length + iv_length:
        d_i = md5(d_i + str.encode(password) + salt).digest()  # obtain the md5 hash value
        d += d_i
    return d[:key_length], d[key_length:key_length + iv_length]

def rsa_encryption(in_file, out_file, public_key, key_length=32):
    bs = AES.block_size  # 16 bytes
    salt = urandom(bs)  # return a string of random bytes
    password = str(random.randint(1000, 10000))
    key, iv = derive_key_and_iv(password, salt, key_length, bs)
    cipher = AES.new(key, AES.MODE_CBC, iv)
    out_file.write(salt)
    finished = False

    while not finished:
        chunk = in_file.read(1024 * bs)
        if len(chunk) == 0 or len(chunk) % bs != 0:  # final block/chunk is padded before encryption
            padding_length = (bs - len(chunk) % bs) or bs
            chunk += str.encode(padding_length * chr(padding_length))
            finished = True
        out_file.write(cipher.encrypt(chunk))
    new_public_key = rsa.PublicKey.load_pkcs1(public_key.encode())
    enc_text = rsa.encrypt(password.encode(), new_public_key)
    print(password)
    return enc_text


2 usages
def rsa_decryption(in_file, out_file, enc_text, private_key, key_length=32):
    new_private_key = rsa.PrivateKey.load_pkcs1(private_key.encode())
    dec_text = rsa.decrypt(enc_text, new_private_key)
    dec_text = dec_text.decode()
    # print('dec_text', dec_text)
    bs = AES.block_size
    salt = in_file.read(bs)
    key, iv = derive_key_and_iv(str(dec_text), salt, key_length, bs)
    cipher = AES.new(key, AES.MODE_CBC, iv)
    next_chunk = ''
    finished = False
    while not finished:
        chunk, next_chunk = next_chunk, cipher.decrypt(in_file.read(1024 * bs))
        if len(next_chunk) == 0:
            padding_length = chunk[-1]
            chunk = chunk[:-padding_length]
            finished = True
        out_file.write(bytes(x for x in chunk))
```

Output:


Ensuring Privacy in the Digital Era:ISDE with Access Control
Admin Login
Data Owner Login
Data User Login
Admin Login Form
Admin ID:
Admin Password :
Login