

Configuration Manual

MSc Research Project
Cloud Computing

Anay Desai
Student ID: X23210125

School of Computing
National College of Ireland

Supervisor: Jorge Mario Cortes Mendoza

**National College of Ireland
Project Submission Sheet
School of Computing**



Student Name:	Anay Desai
Student ID:	X23210125
Programme:	Cloud Computing
Year:	2024
Module:	MSc Research Project
Supervisor:	Jorge Mario Cortes Mendoza
Submission Due Date:	29/01/2025
Project Title:	Configuration Manual
Word Count:	800
Page Count:	8

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:	
Date:	29 January, 2025

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	<input type="checkbox"/>
Attach a Moodle submission receipt of the online project submission , to each project (including multiple copies).	<input type="checkbox"/>
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.	<input type="checkbox"/>

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

Anay Desai
X23210125

1 Introduction

This document manual describes requirement, deployment instructions for Data security using hybrid cryptography approach of ChaCha20-Poly1305 and AES mechanism on Aws and Azure Cloud, Also providing direction for the methodical procedures needed to properly create, operate, test, or reproduce the project. The remaining sections of the whole document are divided into the following sections Module 2 specifies the configuration of the system, Module 3 Libraries needed, Module 4 Database Tables, Module 5 Implementation of Hybrid Encryption technique, Module 6 Cloud Deployment.

2 System Configuration

These are the minimum hardware requirements for the system setup and execution.

2.1 Hardware Configuration

Hardware	Details
Processor	AMD Ryzen 5 3550H with Radeon Vega Mobile Gfx 2.10 GHz
RAM	16 GB DDR4
OS	Windows 10 Pro 64-bit
System	Laptop
Hard Disk	1 TB HDD

Table 1: System Specifications

2.2 Software Configuration

Software	Version/Name
Programming Lang.	Python 3.12.1
Other Software	Visual Studio Code 1.95.3
Encryption types	AES+ChaCha20-Poly1305

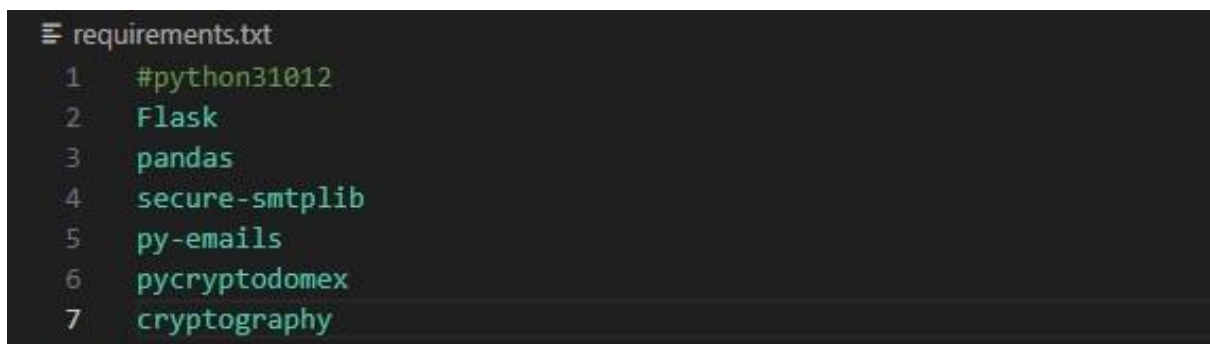
Table 2: System Specifications

3 Environment Setup

3.1 Installing Visual Studio 2022

1. Download Visual Studio
 - (a) Visit the official Visual Studio download page: [Visual Studio Download Page](#).
 - (b) Click on download Visual Studio.
2. Install Visual Studio
 - (a) Run the downloaded installer.
 - (b) Select workloads based on this project.
 - (c) Click install and wait for the process to complete.

The environment setup starts of with the requirements.txt file which contains all the libraries present for the script to run properly.

A screenshot of a code editor with a dark background. The file name 'requirements.txt' is shown in the top left corner. The file contains seven lines of code, each on a new line and numbered from 1 to 7. The code lists Python dependencies: '#python31012', 'Flask', 'pandas', 'secure-smtplib', 'py-emails', 'pycryptodomex', and 'cryptography'.

```
requirements.txt
1 #python31012
2 Flask
3 pandas
4 secure-smtplib
5 py-emails
6 pycryptodomex
7 cryptography
```

Figure 1: Requirements.txt

The implementation starts with importing all the libraries in python to do the hashing and encryption for us, hence are imported at the start of the project.

```
utils.py x requirements.txt
utils.py > ...
1  import os
2  import re
3  import base64
4  import smtplib
5  import sqlite3
6  import warnings
7  import datetime
8  import numpy as np
9  import pandas as pd
10 from email.message import Message
11 from Cryptodome.Cipher import AES
12 from datetime import timedelta, date
13 from email.mime.text import MIMEText
14 from Cryptodome.Random import get_random_bytes
15 from Cryptodome.Util.RFC1751 import key_to_english, english_to_key
16 from cryptography.hazmat.primitives.ciphers.aead import ChaCha20Poly1305
17 warnings.filterwarnings("ignore", category=UserWarning)
18
19 ###===== function =====
20
```

Figure 2: Libraries listed

4 Implementation

After opening the folder and navigating to the application.py file to find for the new users to signup or login.

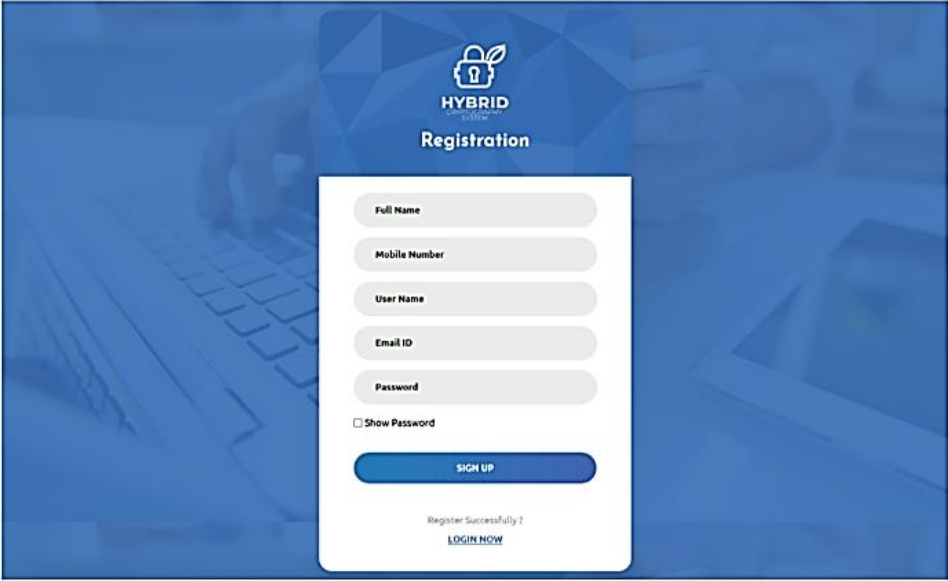
The image shows a registration form for an application named "HYBRID". The form is centered on a blue background with a faint image of a hand typing on a keyboard. The form has a white background and a rounded rectangle shape. At the top of the form is the "HYBRID" logo, which consists of a stylized padlock with a leaf inside, and the text "HYBRID" below it. Below the logo is the word "Registration" in a bold, sans-serif font. The form contains several input fields: "Full Name", "Mobile Number", "User Name", "Email ID", and "Password". Each field has a light gray border and a small icon on the right side. Below the "Password" field is a checkbox labeled "Show Password". At the bottom of the form is a blue button with the text "SIGN UP" in white. Below the button is a link that says "Register Successfully ? LOGIN NOW".

Figure 3: Login Credentials of the User

Post login a sqlite database is created for the user credentials and encrypted data to be stored.

```
#database connection (sqlite)
#create sqlite database file
if os.path.exists('hybrid_cryptography_system.db'):
    pass
else:
    create_db_table()

mydb = sqlite3.connect("hybrid_cryptography_system.db", check_same_thread=False)
mydb.row_factory = sqlite3.Row
mycursor = mydb.cursor()
```

Figure 4: Database

On registering, the user is redirected to the input data page, where the user can input their sensitive data and the algorithm encrypts the data and the user receives the decryption key via email.

The Encryption logic used is AES and ChaCha20. The AES is leveraged with the CFB mode and a static vector the data is re-encrypted the AES ciphertext using a static key of ChaCha20 and a nonce.

The final product is encoded with ChaCha20 ciphertext and the AES key.

The system also uses a CI/CD pipeline to streamline deployment and development in AWS and by also leveraging Codebuild, CodeRun and CodeDeploy.

On the other hand the system is also deployed on Azure for better performance comparisons.

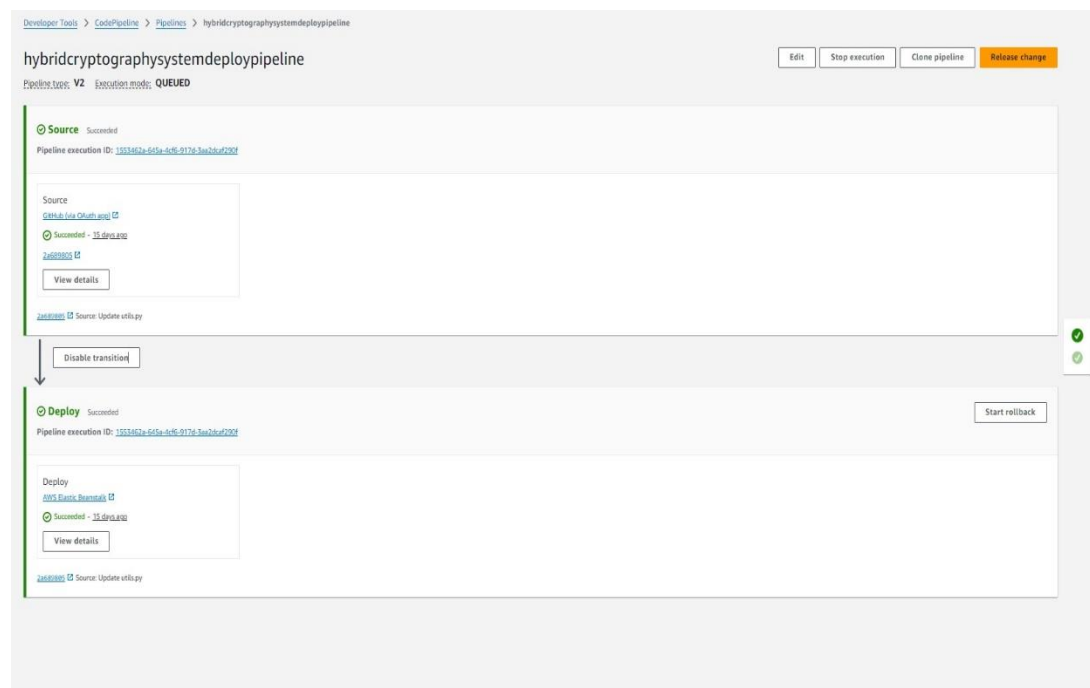


Figure 5: Working Pipeline of the Project in AWS

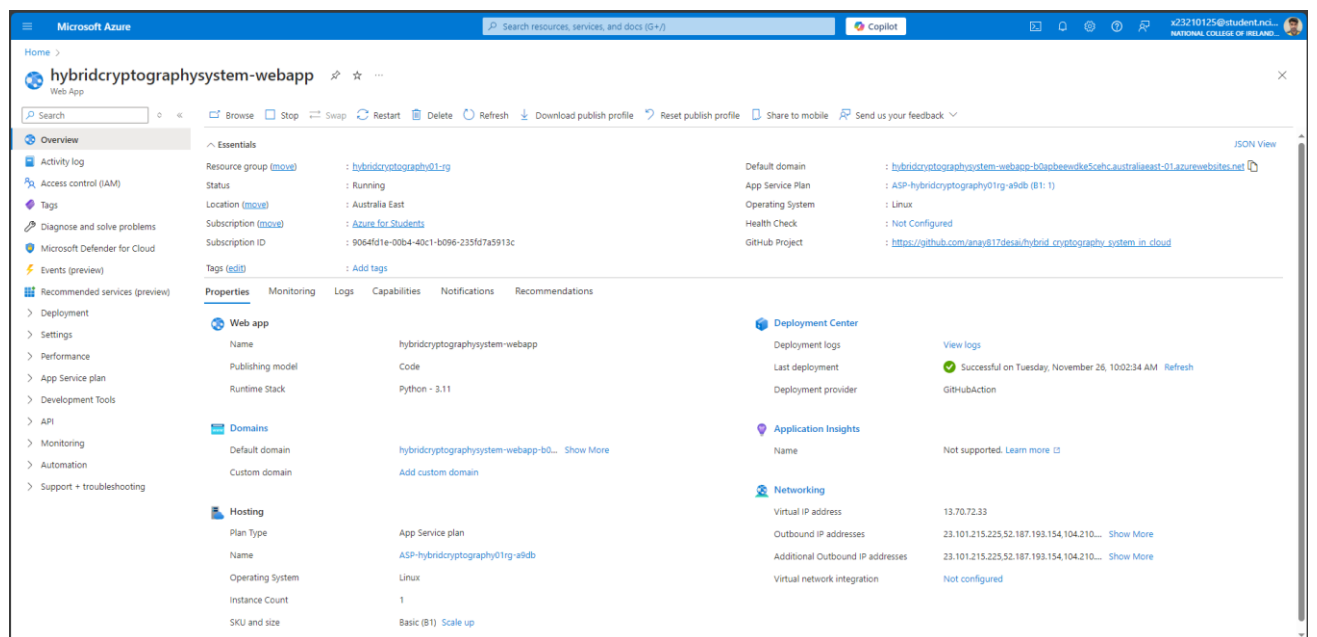


Figure 6: Azure Deployment

5 Evaluation

The results are based on the user input data and randomized to encrypt the data. the performance metrics calculated in this are the avalanche score and encryption and de encryption times. The table below shows the encryption and decryption times in both the clouds.



Figure 7: User adding his healthcare data on web portal

```
return redirect(url_for('login'))
else:
    try:
        msg = session['username']
        user_id = session['u_id']
        age = str(request.form['age'])
        trestbps = str(request.form['trestbps'])
        cholesterol = str(request.form['cholesterol'])
        thalach = str(request.form['thalach'])
        curr_date_time = currdatetime()
        encstart = time.time() #enc time
        org_key = get_random_bytes(16)
        #encryption
        ciphertext_age, key = aeschaencryption(org_key, age)
        ciphertext_trestbps, key = aeschaencryption(org_key, trestbps)
        ciphertext_cholesterol, key = aeschaencryption(org_key, cholesterol)
        ciphertext_thalach, key = aeschaencryption(org_key, thalach)
        #insert enc data
        sql="INSERT INTO Health_Data VALUES (NULL, ?, ?, ?, ?, ?, ?, ?, ?, ?)"
        mycursor.execute(sql, (user_id, curr_date_time, ciphertext_age, ciphertext_trestbps, ciphertext_cholesterol, ciphertext_thalach, key, age, trestbps, cholesterol, thalach))
        mydb.commit()
        successmsg = "Data Added Successfully"
        encend = time.time()

        sql="SELECT emailid FROM User_Master WHERE u_id = ?"
        mycursor.execute(sql, (user_id,))
        email_database = mycursor.fetchone()
        email_id = email_database['emailid']
        subject = "Encryption/Decryption Key of Health Data"
        textmsg = "Your Data Added Successfully - Datetime: {} & Key: {}".format(curr_date_time, key)
        send_email(textmsg, subject, email_id)

        encstart = encend - encstart
        encstart = encstart * 1000
        encrypt_time = "Encryption Time: {} ms".format(int(encstart))
        return render_template('adddata.html', msg = msg, successmsg = successmsg, encrypt_time=encrypt_time)
```

Figure 8: Encryption Scheme


```
def aeschachaencryption(key, inpdata):
    #aes
    inp1 = bytes(inpdata, 'utf-8')
    iv = b'\x1f_\x7f\xb1\xb9\xb6\x0e\xe2oI5E\x82\xac:G'
    cipher = AES.new(key, AES.MODE_CFB, iv=iv)
    ciphertext = cipher.encrypt(inp1)
    ciphertext = base64.b64encode(ciphertext).decode()
    key = base64.b64encode(key).decode()
    #chacha
    data = ciphertext
    data = data.encode("utf-8")
    aad = b"authenticated but unencrypted data"
    chachakey = b'\xf6\xf5@\xbf\x85\xac)\xe5\x02\xb2\xde\xe6\x8f\x8e\x1b\xc6?\xe4\xad4|j\xe1\xc6J=\xec\xcaif\xbd'
    chacha = ChaCha20Poly1305(chachakey)
    nonce = b'|\xca\xd4\x15\x8c\x02$\xcc\xa4<\xe1\x9f'
    ct = chacha.encrypt(nonce, data, aad)
    ct = base64.b64encode(ct).decode()
    return ct, key
```

Figure 9: Encryption Process

```
def aeschachadecryption(ciphertext, key):
    #chacha
    ct = base64.b64decode(ciphertext)
    chachakey = b'\xf6\xf5@\xbf\x85\xac)\xe5\x02\xb2\xde\xe6\x8f\x8e\x1b\xc6?\xe4\xad4|j\xe1\xc6J=\xec\xcaif\xbd'
    nonce = b'|\xca\xd4\x15\x8c\x02$\xcc\xa4<\xe1\x9f'
    aad = b"authenticated but unencrypted data"
    chacha = ChaCha20Poly1305(chachakey)
    decrypt_output = chacha.decrypt(nonce, ct, aad)
    decrypt_output = decrypt_output.decode("utf-8")
    #print(decrypt_output)
    aad = b"authenticated but unencrypted data"
    #aes
    decodemsg = base64.b64decode(decrypt_output)
    key = base64.b64decode(key)
    iv = b'\x1f_\x7f\xb1\xb9\xb6\x0e\xe2oI5E\x82\xac:G'
    cipher = AES.new(key, AES.MODE_CFB, iv=iv)
    decrypt_data = cipher.decrypt(decodemsg)
    decrypt_data = decrypt_data.decode("utf-8")
    return decrypt_data
```

Figure 10: Decryption Scheme

The Decryption page looks something like this, while the backend performs process and decrypts the data for the user.

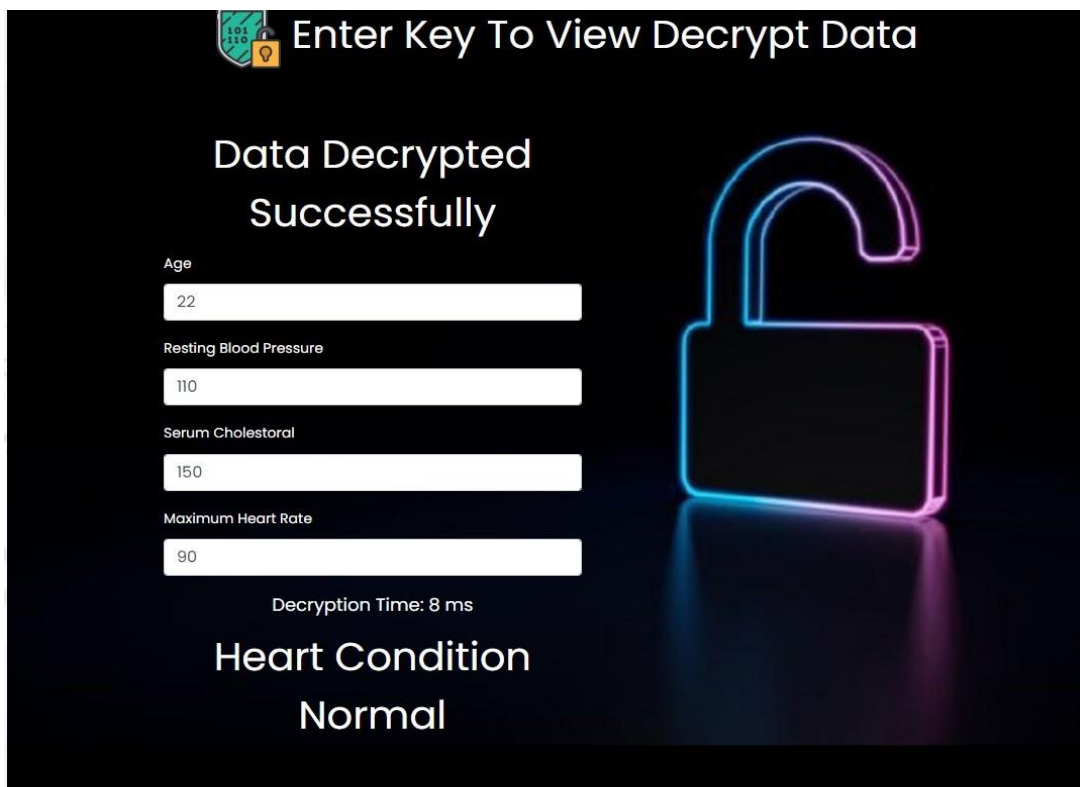


Figure 11: Decryption Process

The encryption and decryption times in the cloud are as follows;

Encryption Time	Decryption Time
1.5	1.32
1.95	2.01
1.56	1.78
1.78	1.6

Figure 12: Time metric in AWS

Encryption Time	Decryption Time
1.52	1.6
1.05	2.0
1.87	2.2
1.6	1.87

Figure 13: Time metric in Azure

The avalanche score is also calculated to test the strength of the algorithm.

The decryption key when uploaded and successfully reveals the data, the decryption time is shown along with the user's health Status. The script for calculating the score is

```
16
17 binary1 = '/UQCorC+w4WBZ3hSO1MPKA=='
18 binary2 = '8GeHbrAA3sOKJ6xU5lM94g=='
19
20
21 avalanche_score = calculate_avalanche_score(binary1, binary2)
22 print('Avalanche Score:', avalanche_score)
23
```

PROBLEMS OUTPUT TERMINAL PORTS SEARCH ERROR DEBUG CONSOLE

WebApplication>C:/Users/Administrator/anaconda3/envs/hybridcryptographynv/python.exe c:/Users/Administrator/
ers/Administrator/Desktop/hybrid_cryptography_system_in_cloud/WebApplication/avalanche_eff
ct.py
Avalanche Score: 97.82608695652173 WebApplication>

Figure 14: Avalanche Score