National College of Ireland

Configuration Manual

MSc Research Project
Cloud Computing

Devaswaroopa Machenhalli Nandish
Student ID: 22169245

School of Computing
National College of Ireland

Supervisor: Dr Ahmed Makki

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Devaswaroopa Machenhalli Nandish
Student ID:	22169245
Programme:	Cloud Computing
Year:	2023
Module:	MSc Research Project
Supervisor:	Dr Ahmed Makki
Submission Due Date:	14/12/2023
Project Title:	Configuration Manual
Word Count:	382
Page Count:	11

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:	Devaswaroopa Machenahalli Nandish
Date:	15th December 2023

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

Devaswaroopa Machenhalli Nandish
22169245

1 Introduction

The configuration manual demonstrates running of the integrated code. The code should be properly configured as aligned in the configuration manual to successfully observe the results.

1.1 Requirement

Before commencing the setup process, verify that the following requirements have been fulfilled:

- Possession of data in CSV format.
- An active AWS account.
- Access to SageMaker with Compute-Optimized instances.
- Proficiency in Boto 3 and Python.
- Install Pycharm , NymPy, argon2-cffi, psutil, and matplotlib libraries
- Optionally, contemplate the use of VScode for future tasks.

2 Dependency Installations

1. Install "pip install package-name" command
2. Install "Pip Install pandas"
3. Install "pip install matplotlib"

3 Implementation

1. Log in to your AWS Account and Go to AWS S3 buckets

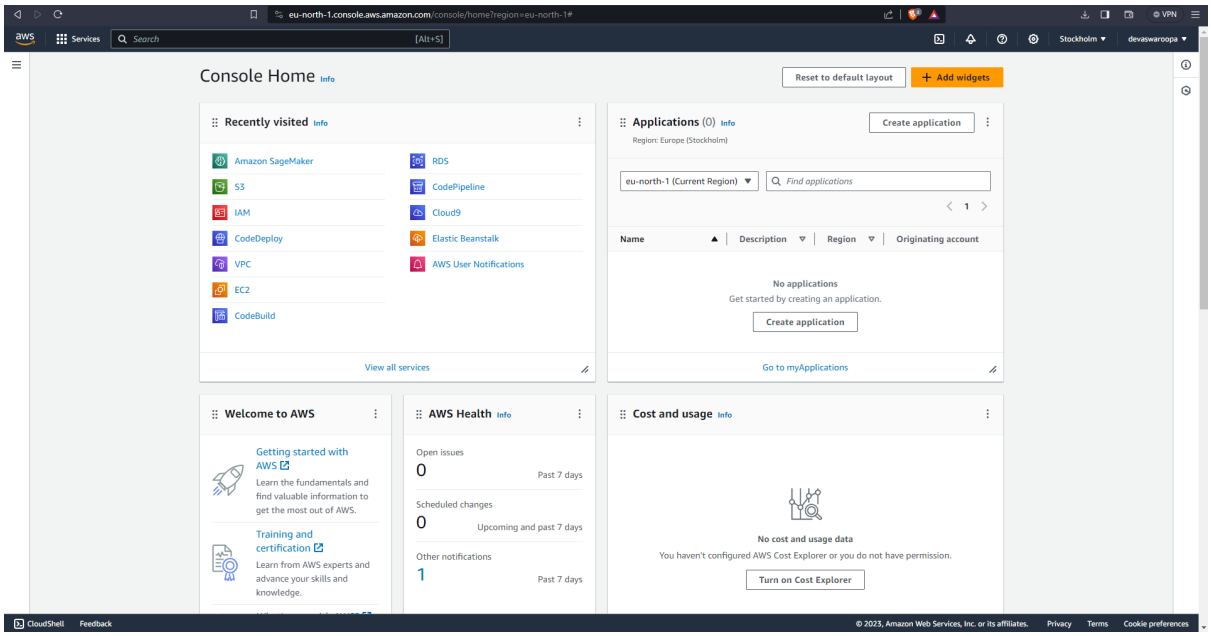

Figure 1: AWS Console Page

2. Next step Create S3 Bucket for storing csv files.

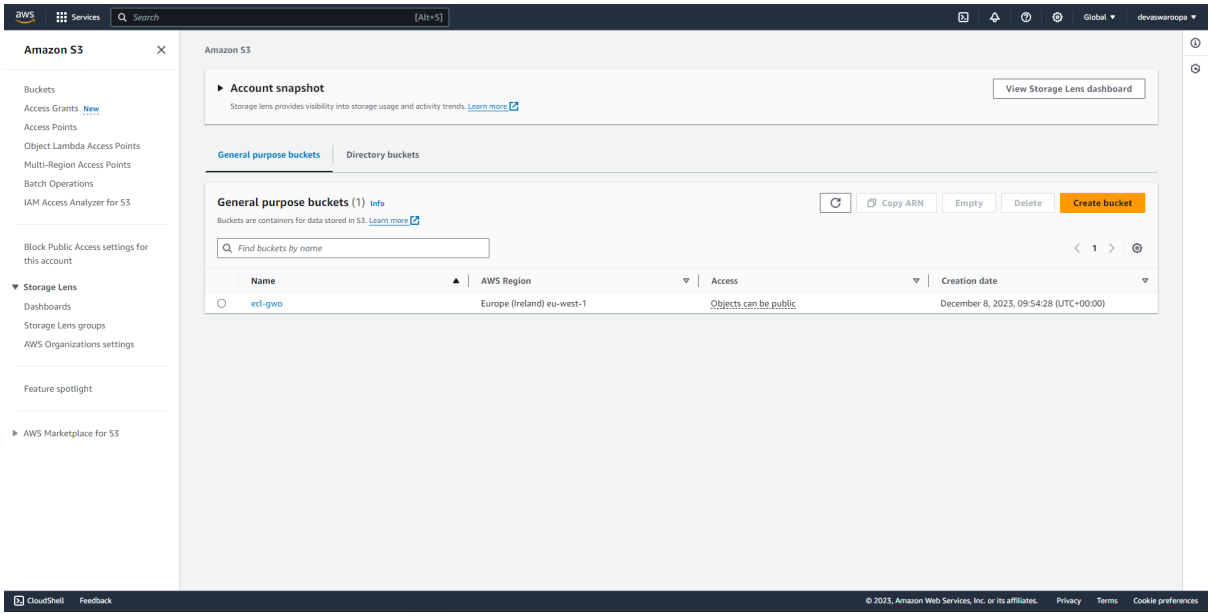

Figure 2: AWS S3 Bucket

3. Upload the CSV files to the s3 bucket.

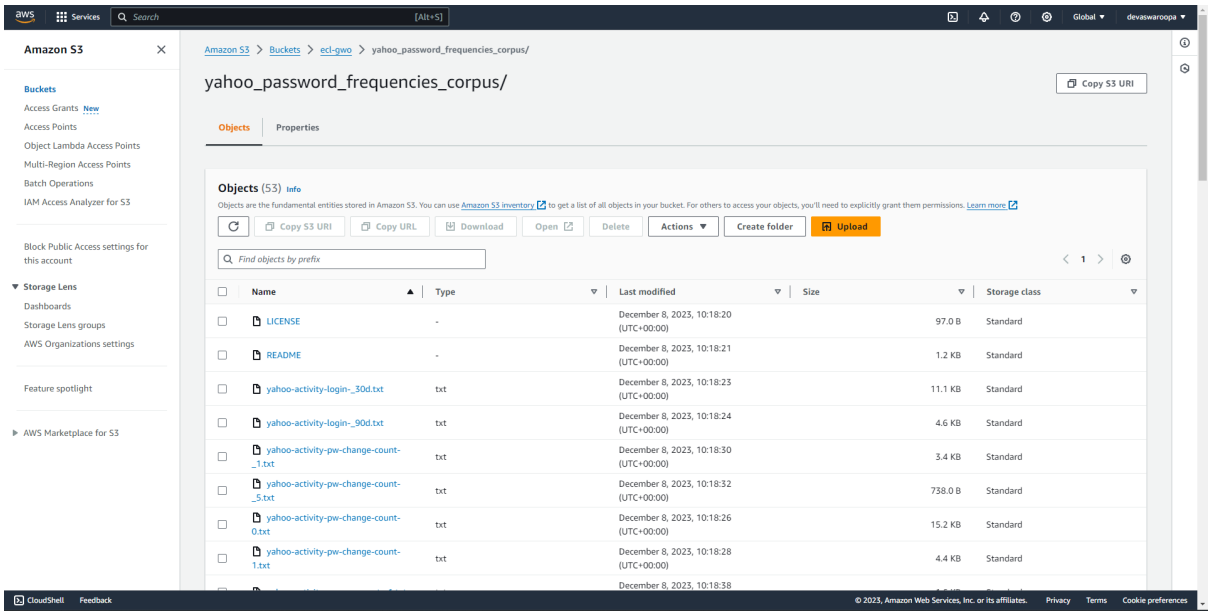

Figure 3: Adding CSV files to S3 Bucket

4. Setup Policies for User

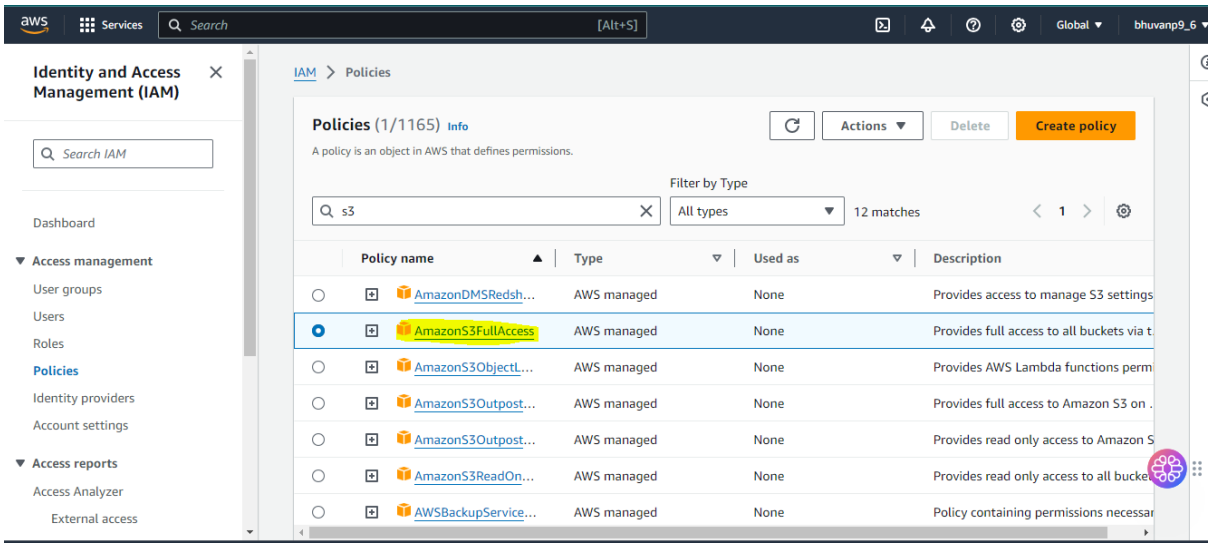

Figure 4: Set Policies to S3 bucket

5. Create notebook instance to upload the .ipynb and dataset files as shown

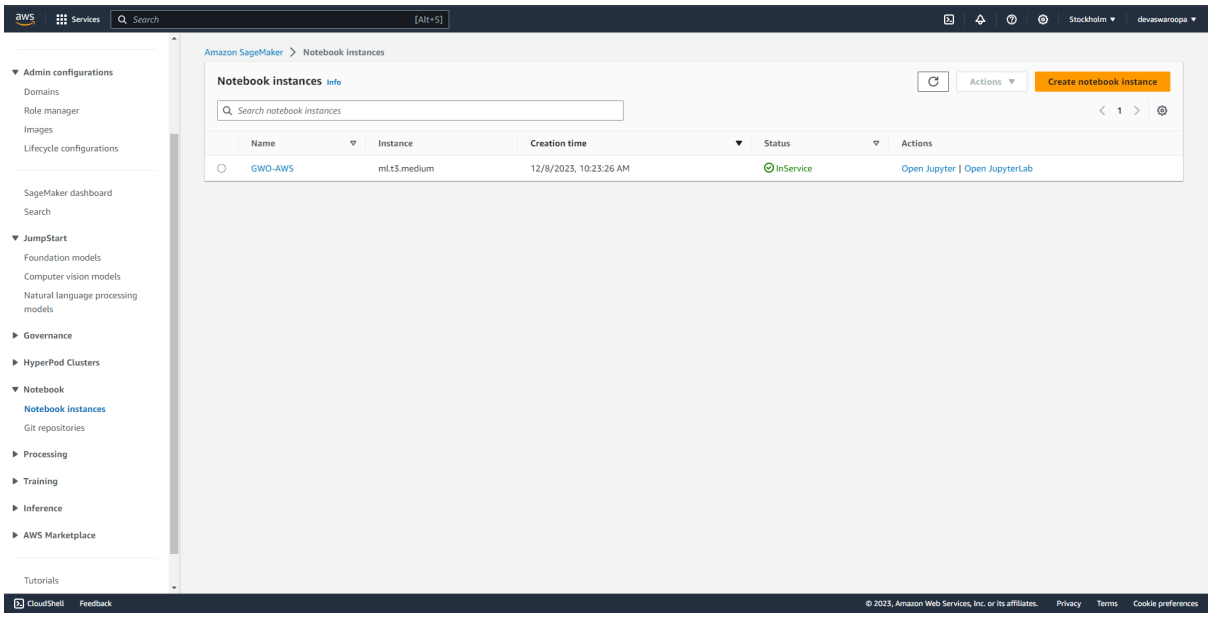

Figure 5: Jpyter Notebook

4 Python Libraries

1. Pip Install all the dependencies
2. Let us import the necessary libraries.

```
In [28]: import os
         from cryptography.hazmat.backends import default_backend
         from cryptography.hazmat.primitives import hashes
         from cryptography.hazmat.primitives.kdf.pbkdf2 import PBKDF2HMAC
         from cryptography.hazmat.primitives.ciphers import Cipher, algorithms, modes
         from argon2 import PasswordHasher
         import numpy as np
         import time
         import psutil
         import matplotlib.pyplot as plt
```

Figure 6: Import the python library

3. Run the PBKDF2 and Argon2 notebook after configuring our notebook.

```
pbk_df2 = []
arg_on2 = []
cpu_usage_list = []
memory_usage_list = []

# Load password dataset
def load_password_dataset(file_path):
    with open(file_path, 'r') as file:
        lines = file.readlines()
        passwords = [line.split()[0] for line in lines]
    return passwords

# Step 3: Create cryptographic keys using PBKDF2 and Argon2

def generate_pbkdf2_key(password, salt, iterations=100000):
    kdf = PBKDF2HMAC(
        algorithm=hashes.SHA256(),
        iterations=iterations,
        length=32,
        salt=salt,
        backend=default_backend()
    )
    key = kdf.derive(password.encode())
    return key

def generate_argon2_key(password):
    ph = PasswordHasher()
    key = ph.hash(password)
    return key.encode()  # Convert the key to bytes

# Step 4: Implement encryption method and key generation

def encrypt_data(data, key):
    iv = os.urandom(16)
    cipher = Cipher(algorithms.AES(key[:32]), modes.CFB(iv), backend=default_backend())
    encryptor = cipher.encryptor()
    ciphertext = encryptor.update(data) + encryptor.finalize()
    return ciphertext, iv

def decrypt_data(ciphertext, key, iv):
    cipher = Cipher(algorithms.AES(key[:32]), modes.CFB(iv), backend=default_backend())
    decryptor = cipher.decryptor()
    plaintext = decryptor.update(ciphertext) + decryptor.finalize()
    return plaintext
```

Figure 7: PBKDF2 and Argon2

4. Visualisation of the reuslts of PBKDF2 And Argon2

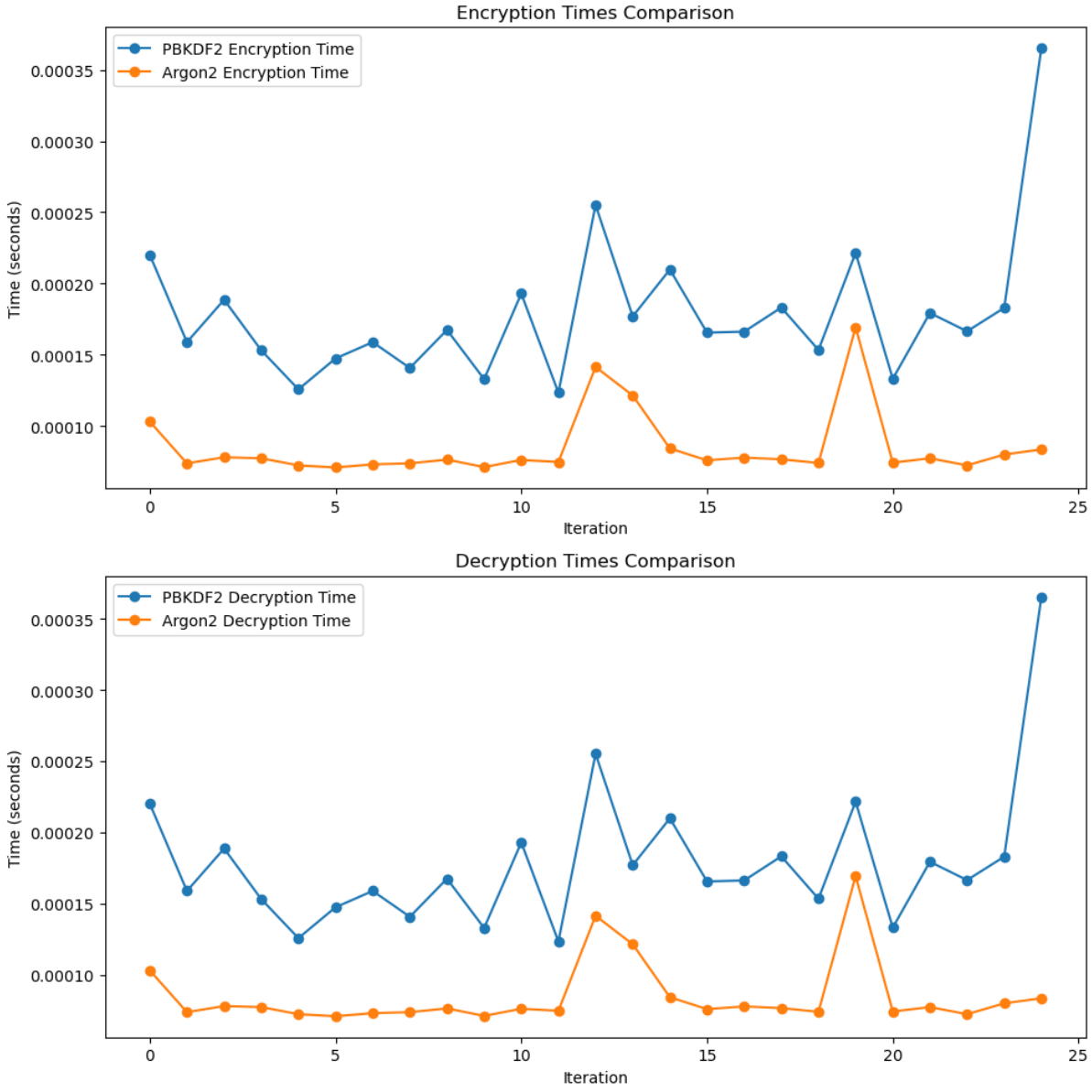

Figure 8: PBKDF2 & Argon2 - Encrypt & Decrypt.

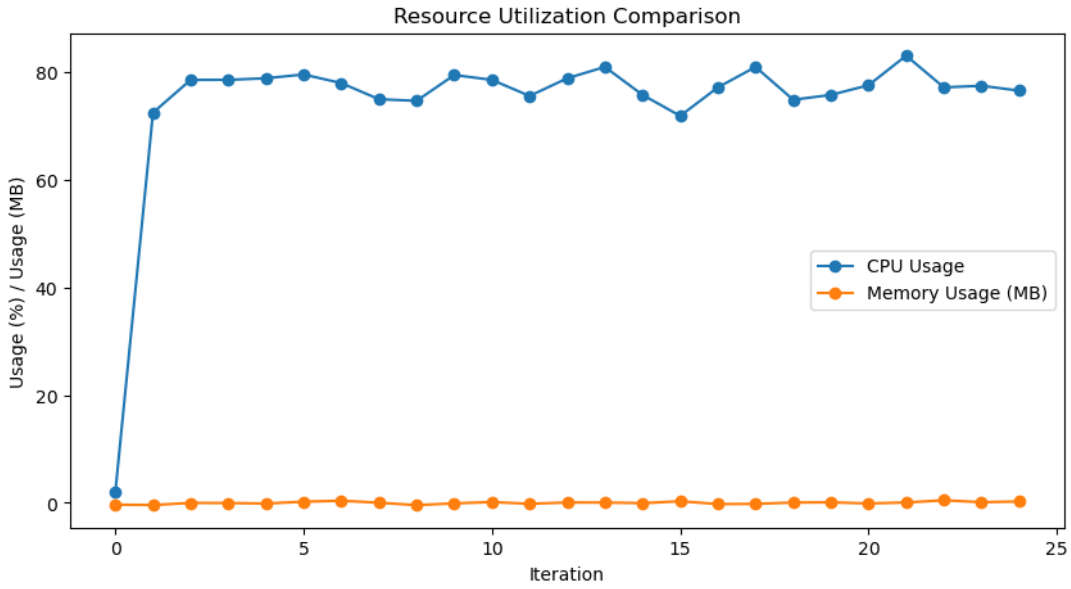

Figure 9: PBKDF2 & Argon2 - CPU&Memory Utilization.

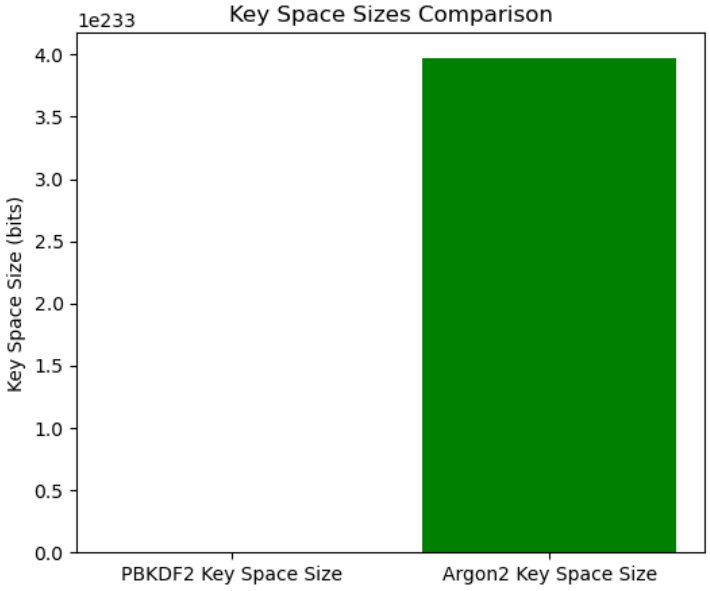

Figure 10: PBKDF2 & Argon2 - Key Space.

5. Run the ECL-GWO with key generated by PBKDF2/Argon2 keys notebook after configuring our notebook.

```python
# Step 3: Create cryptographic keys using PBKDF2 and Argon2

def generate_pbkdf2_key(password, salt):
    if isinstance(password, str):
        password = password.encode()  # Convert to bytes if it's a string

    kdf = PBKDF2HMAC(
        algorithm=hashes.SHA256(),
        iterations=100000,  # You can adjust the number of iterations
        length=32,
        salt=salt,
        backend=default_backend()
    )
    key = kdf.derive(password)
    return key

def generate_argon2_key(password):
    ph = PasswordHasher()
    key = ph.hash(password)
    return key.encode()  # Convert the key to bytes

# Enhanced Cryptographic Layer with Grey Wolf Optimization (ECL-GWO) key generation
def ecl_gwo_key_generation(password, salt):
    # Grey Wolf Optimization (GWO) algorithm for key modification
    def gwo_algorithm(key):
        # For GWO Modification
        modified_key = bytearray(key) + bytearray(os.urandom(len(key)))  # ECL-GWO modification
        return bytes(modified_key)

    # Generate the original key using PBKDF2
    original_key = generate_pbkdf2_key(password, salt)

    # Apply Grey Wolf Optimization (GWO) to enhance the key
    enhanced_key = gwo_algorithm(original_key)

    return enhanced_key

# Implement encryption method with ECL-GWO key
def encrypt_data(data, key):
    iv = os.urandom(16)

    # Apply ECL-GWO enhanced key to the encryption process
    ecl_gwo_key = ecl_gwo_key_generation(key, iv)
    cipher = Cipher(algorithms.AES(ecl_gwo_key[:32]), modes.CFB(iv), backend=default_backend())
```

Figure 11: ECL-GWO Code with PBKDF2/Argon 2 Keys in notebook

6. Visualisation of the reuslts of ECL-GWO with PBKDF2/Argon 2 keys

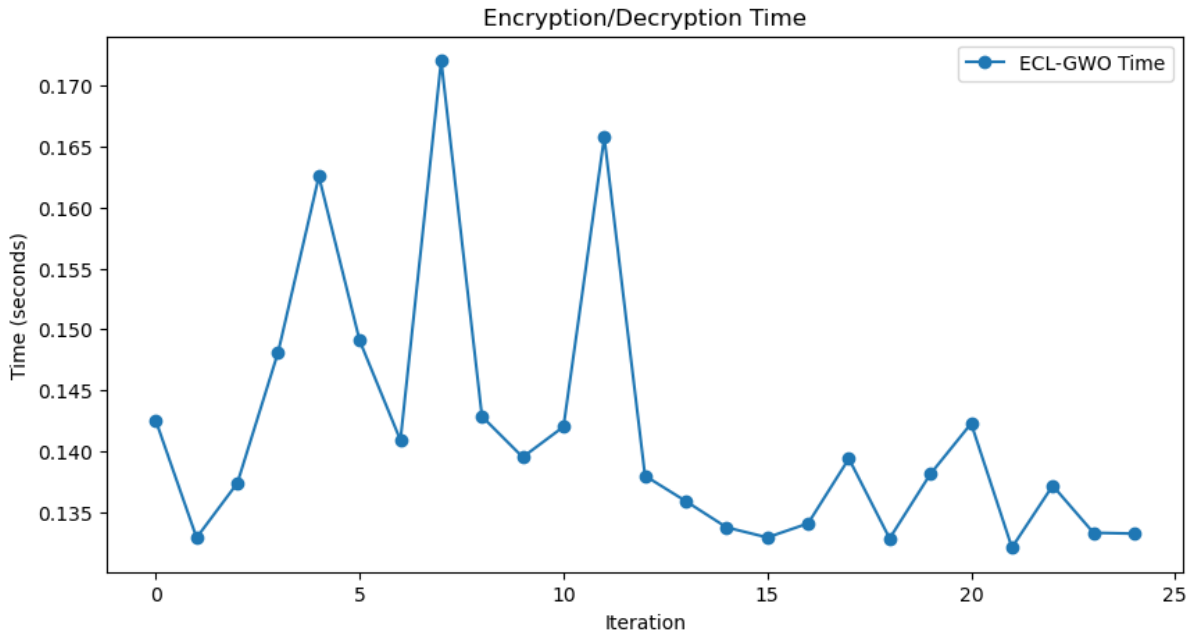

Figure 12: ECL-GWO PBDFK2 Encrypt & Decrypt Time

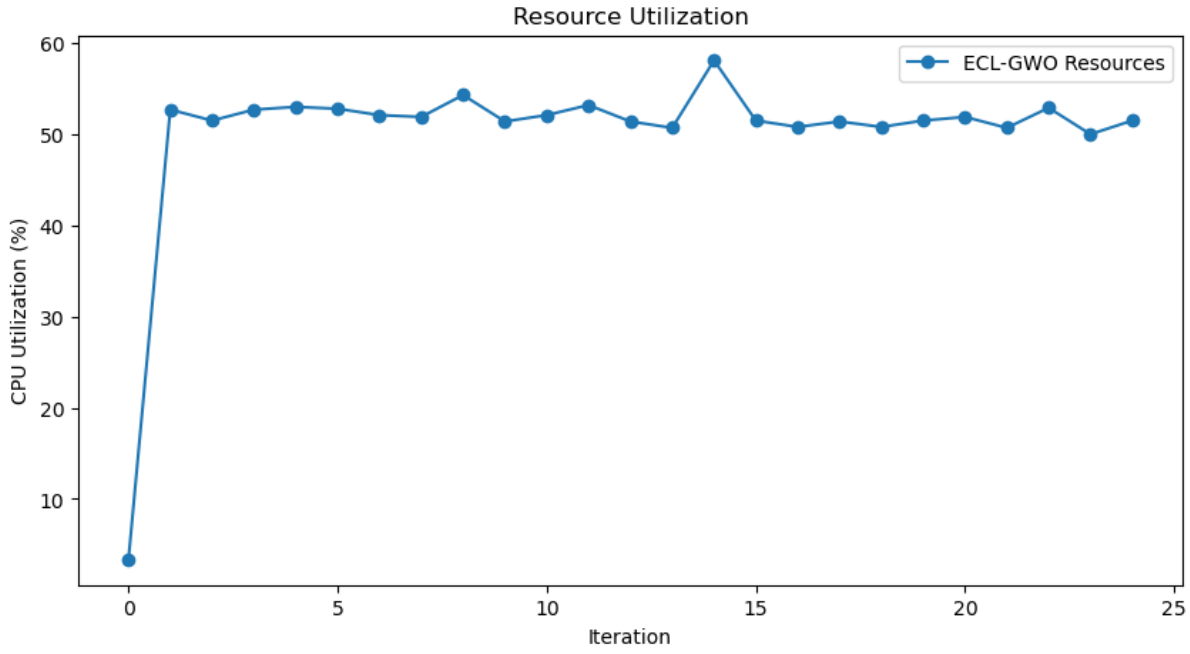

Figure 13: ECL-GWO PBDFK2 - CPU&Memory Utilization.

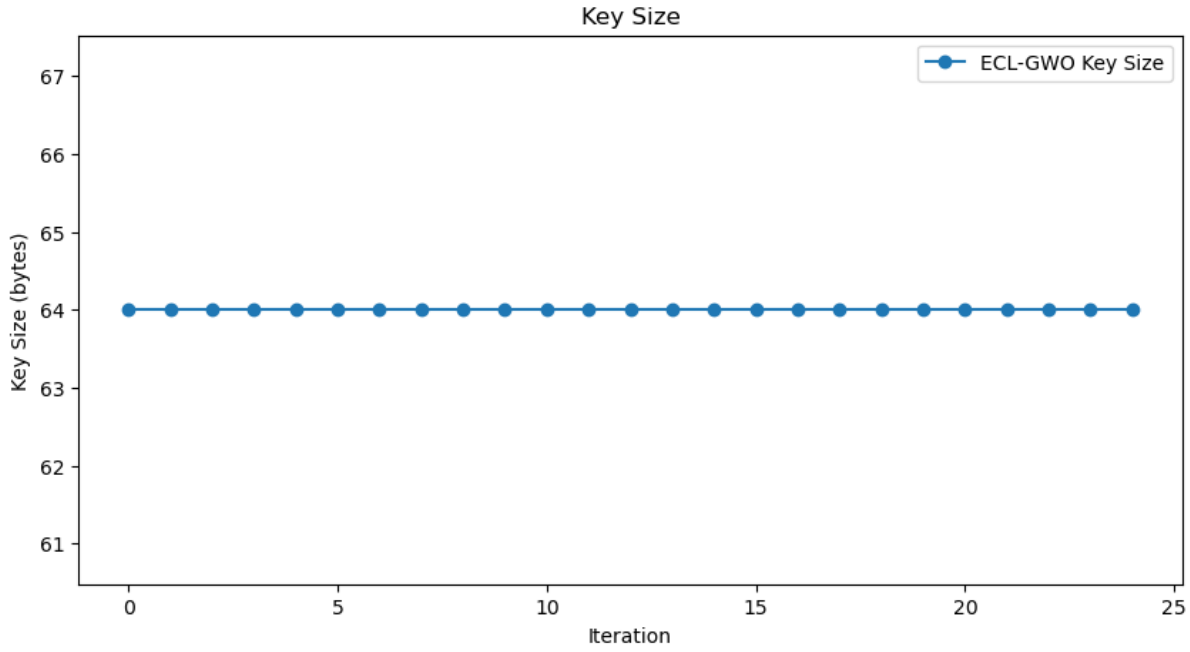

Figure 14: ECL-GWO along with PBKDF2 Key Generation size.

7. Run the ECL-GWO with key generated independently notebook after configuring our notebook.

```python
# Step 3: Create cryptographic keys using PBKDF2 and Argon2

def generate_pbkdf2_key(password, salt):
    if isinstance(password, str):
        password = password.encode()  # Convert to bytes if it's a string

    kdf = PBKDF2HMAC(
        algorithm=hashes.SHA256(),
        iterations=100000,  # You can adjust the number of iterations
        length=32,
        salt=salt,
        backend=default_backend()
    )
    key = kdf.derive(password)
    return key

def generate_argon2_key(password):
    ph = PasswordHasher()
    key = ph.hash(password)
    return key.encode()  # Convert the key to bytes

# Enhanced Cryptographic Layer with Grey Wolf Optimization (ECL-GWO) key generation
def ecl_gwo_key_generation(password, salt):
    # Grey Wolf Optimization (GWO) algorithm for key modification
    def gwo_algorithm(key):
        # Assuming GWO modifies the key
        modified_key = bytearray(key) + bytearray(os.urandom(len(key)))
        return bytes(modified_key)

    # Generate the original key using PBKDF2
    original_key = generate_pbkdf2_key(password, salt)

    # Apply Grey Wolf Optimization (GWO) to enhance the key
    enhanced_key = gwo_algorithm(original_key)

    return enhanced_key
```

Figure 15: ECL-GWO Code with Indenepend key Generation in notebook

8. Visualisation of the reuslts of ECL-GWO with key generated independently notebook after configuring our notebook.

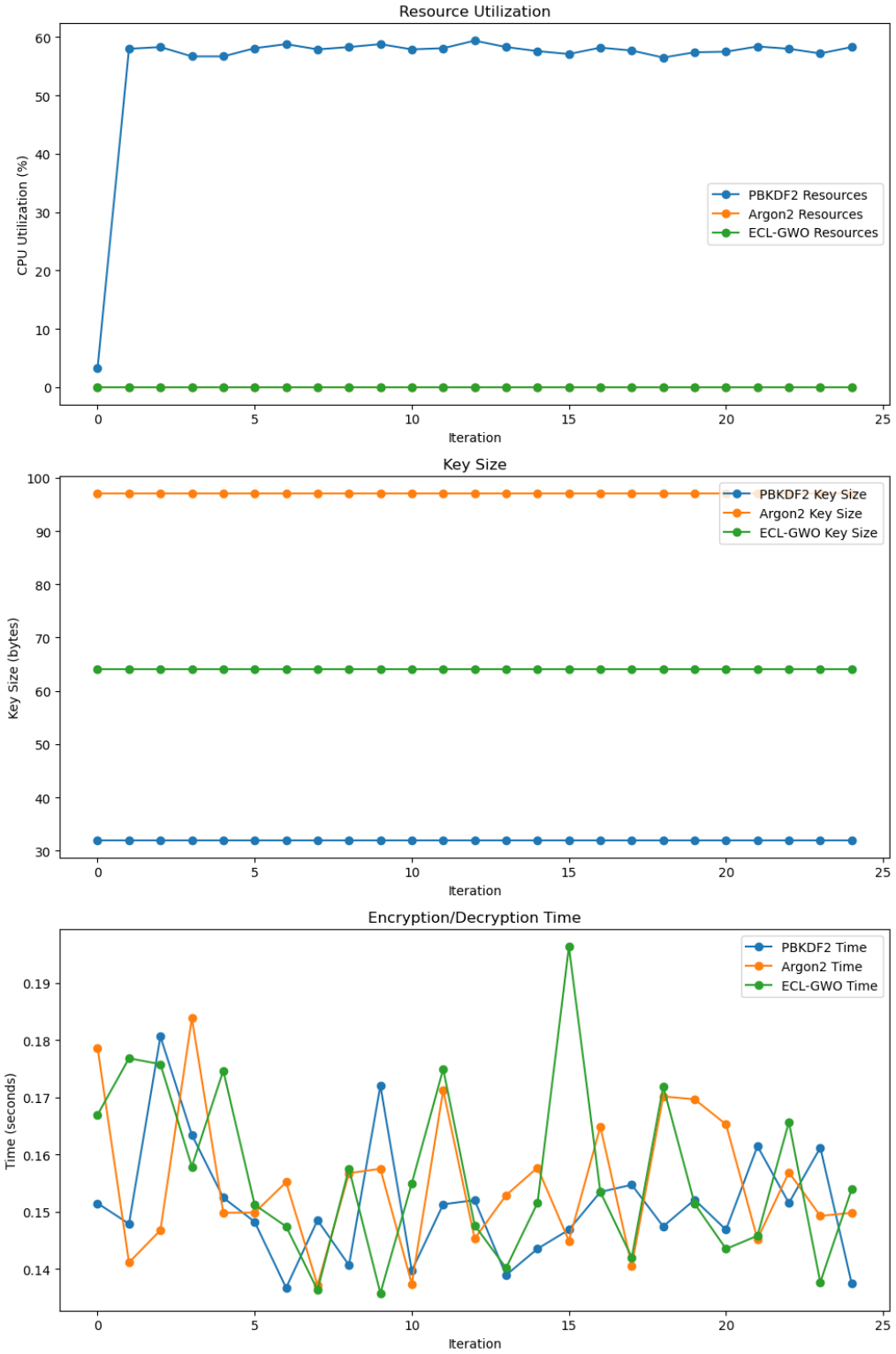

Figure 16: ECL-GWO along with PBKDF2 and Argon2 with independent Key Generation. .