Phase 2: Privacy and Security Implementation Report

1. Introduction

This report documents the implementation of privacy policies and protection procedures for safeguarding the confidentiality and integrity of our dataset, in compliance with relevant regulations. The following sections cover:

- **Data Security and Encryption** techniques
- **Access Control and Privacy Policies**
- **Regulatory Compliance** using Python libraries

---

2. Data Security and Encryption Implementation

To protect sensitive information, we apply hashing and substitution encryption algorithms using Python libraries.

2.1 Loading the Dataset

```
import pandas as pd


data = pd.read_csv(r"C:\ZC\Data Governance\DataGovernanceWorkflow\data\Cleaned_csv.csv")

# Drop any residual index column from prior exports

df = pd.DataFrame(data).drop(columns="Unnamed: 0")
```

Explanation:

- `pd.read_csv`: loads the CSV file into a DataFrame.
- `drop(columns="Unnamed: 0")`: removes the unnecessary index column.

2.2 Symmetric Encryption of Passwords

```
# Install the cryptography library if not already installed:
# pip install cryptography
from cryptography.fernet import Fernet

# Generate a symmetric encryption key
key = Fernet.generate_key()
fernet = Fernet(key)

# Encrypt each password in the DataFrame
encrypted_list = []
for pwd in df['Password']:
    token = fernet.encrypt(pwd.encode()).decode()
    encrypted_list.append(token)

# Add the encrypted passwords as a new column
df['Password_encrypted'] = encrypted_list
print(df['Password_encrypted'].head())
```

Explanation:

- `Fernet.generate_key()`: creates a new secure key.

- `fernet.encrypt(...)`: encrypts each password and returns a base64 string.

- Encrypted values are stored in `Password_encrypted`.

2.3 Verifying Decryption

```
# Decrypt the encrypted passwords to verify correctness
decrypted_list = []
for token in df['Password_encrypted']:
    original = fernet.decrypt(token.encode()).decode()
    decrypted_list.append(original)

print(decrypted_list[:5])  # Display first five decrypted passwords
```

Explanation:

- `fernet.decrypt(...)`: restores each original password, confirming encryption/decryption integrity.

2.4 Substitution Encryption: Caesar Cipher

```
def caesar_cipher(text: str, shift: int) -> str:
    """
    Applies a Caesar cipher shift to alphabetic characters in `text`.
    Non-alphabet characters remain unchanged.
    """
    result = ''
    for ch in text:
        if ch.isupper():
            result += chr((ord(ch) - 65 + shift) % 26 + 65)
        elif ch.islower():
            result += chr((ord(ch) - 97 + shift) % 26 + 97)
        else:
            result += ch
    return result


# Encrypt textual columns with a shift of 3
df['Username_encrypted'] = df['Username'].apply(lambda x: caesar_cipher(x, 3))
df['Country_encrypted']  = df['Country'].apply(lambda x: caesar_cipher(x, 3))
df['City_encrypted']     = df['City'].apply(lambda x: caesar_cipher(x, 3))
```

Explanation:

- `caesar_cipher`: shifts each letter by the specified key (3) to encrypt the text.
- Applied to `Username`, `Country`, and `City` fields.

---

3. Access Control and Privacy Policies

Role-Based Access Control (RBAC) ensures that only authorized users have specific permissions.

3.1 Defining Roles and Permissions

```
import random
from collections import defaultdict
```

```
# Define available roles and initialize user lists
roles = {
    'system': [], 'services': [], 'game': [], 'temporary': [],
    'analytics': [], 'monitoring': [], 'support': [],
    'devops': [], 'network': [], 'admin': []
}
# Map each role to its allowed operations
permissions = {
    'system':     ['read', 'write', 'delete', 'update'],
    'services':   ['read', 'write'],
    'game':       ['read', 'write', 'update'],
    'temporary':  ['read'],
    'analytics':  ['read', 'write'],
    'monitoring': ['read'],
    'support':    ['read', 'write'],
    'devops':     ['read', 'write', 'delete'],
    'network':    ['read', 'write'],
    'admin':      ['read', 'write', 'delete']
}

# Randomly assign each unique username to a role
user_list = df['Username'].unique().tolist()
for user in random.sample(user_list, len(user_list)):
    assigned_role = random.choice(list(roles.keys()))
    roles[assigned_role].append(user)


# Helper to get roles for a username

def get_user_roles(username: str) -> list:

    return [r for r, users in roles.items() if username in users] or ['uncategorized']


# Display role assignments

for role, users in roles.items():

    print(f"{role.title()} ({len(users)} users): {users}")
```

Explanation:

- `roles`: maps each role name to its assigned users.
- `permissions`: specifies allowed operations per role.
- Random assignment simulates provisioning; `get_user_roles` retrieves a user's roles.

---

4. Regulatory Compliance Using Python

Demonstrate compliance with GDPR, CCPA, and HIPAA using libraries or fallback stubs.

4.1 Setup and Imports

```
import pandas as pd

import numpy as np

import json


# GDPR library or stub
```

```
try:
    from python_gdpr_utils import GDPRAnonymizer
    anonymizer = GDPRAnonymizer()
    gdpr_lib = True
except ModuleNotFoundError:
    import hashlib
    class GDPRStub:
        @staticmethod
        def anonymize_ip(ip):
            parts = str(ip).split('.')
            return f"{parts[0]}.{parts[1]}.{parts[2]}.0"
        @staticmethod
        def pseudonymize(val):
            h = hashlib.md5(str(val).encode()).hexdigest()
            return f"anon_{h[:8]}"
    anonymizer = GDPRStub()
    gdpr_lib = False

# CCPA library or stub
try:
    from ccpa import CCPAProcessor
    ccpa_processor = CCPAProcessor()
    ccpa_lib = True
except ModuleNotFoundError:
    ccpa_processor = None

    ccpa_lib = False


# HIPAA scanners or stub

try:

    from Hippo import HippoScanner

    import pyTenable, SecurityMonkey

    hippo = HippoScanner()

    tenable = pyTenable.TenableIO()

    sec_monkey = SecurityMonkey()

    hipaa_lib = True

except ModuleNotFoundError:

    hipaa_lib = False
```

Explanation:

- Attempts to import each compliance library; uses stubs when unavailable.

4.2 GDPR Compliance

```
gdpr_columns = ['IP', 'Username', 'Password', 'City', 'Country']

def apply_gdpr(df_input: pd.DataFrame) -> pd.DataFrame:

    df_copy = df_input.copy()

    for col in gdrp_columns:

        if col in df_copy:

            if col == 'IP':

                df_copy[col] = df_copy[col].apply(anonymizer.anonymize_ip)
            else:
                df_copy[col] = df_copy[col].apply(anonymizer.pseudonymize)
    return df_copy


# Generate GDPR-compliant CSV
df_gdpr = apply_gdpr(df)
df_gdpr.to_csv('gdpr_compliant.csv', index=False)
```

Explanation:

- Masks IP addresses by zeroing the last octet.
- Pseudonymizes other personal data via hashing or library methods.

4.3 CCPA Compliance

```
def apply_ccpa(df_input: pd.DataFrame) -> pd.DataFrame:
    df_copy = df_input.copy()
    # Random consistent opt-out flag per user
df_copy['DoNotSell'] = df_copy['Username'].map(
        lambda u: np.random.RandomState(hash(u) % 2**32).choice([True, False])
    )
    df_copy['CanSellData'] = ~df_copy['DoNotSell']
    return df_copy


# Generate CCPA-compliant CSV
df_ccpa = apply_ccpa(df)
df_ccpa.to_csv('ccpa_compliant.csv', index=False)
```

Explanation:

- `DoNotSell` flag determines user’s opt-out preference.
- `CanSellData` indicates whether data sale is permitted.

4.4 HIPAA Compliance

```
def apply_hipaa_audit():
    if not hipaa_lib:
        return {'config_issues': [], 'vulnerabilities': [], 'policy_findings': []}
    return {
        'config_issues': hippo.scan_config('data_path'),
        'vulnerabilities': tenable.scan_asset_group('NetworkLogs'),
        'policy_findings': sec_monkey.audit()
    }


hipaa_report = apply_hipaa_audit()
with open('hipaa_report.json', 'w') as f:
    json.dump(hipaa_report, f, indent=4)
```

Explanation:

- Generates a JSON report of configuration issues, vulnerabilities, and policy findings.

---

5. Summary and Submission

- **GDPR:** Anonymized or pseudonymized PII; saved to `gdpr_compliant.csv`.
- **CCPA:** Added opt-out flags and sale permissions; saved to `ccpa_compliant.csv`.
- **HIPAA:** Generated audit report; saved to `hipaa_report.json`.

Please submit this report along with the source code files via the classroom portal for Phase 2 review.