HealthID Privacy Metrics with Homomorphic Encryption

Bonafide record of work done by

Akshayakumar N G	**21z206**
Anbukkumar P K	**21z208**
Arulpathi A	**21z209**
Gaurav Vishnu N	**21z217**
Abaikumar I	**22z434**

19Z701 - CRYPTOGRAPHY

Dissertation submitted in partial fulfillment of the requirement for the award of degree of

BACHELOR OF ENGINEERING

Branch: COMPUTER SCIENCE AND ENGINEERING
Of Anna University

OCT 2024

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

PSG COLLEGE OF TECHNOLOGY

(Autonomous Institution)

COIMBATORE – 641 004

Table Of Contents

Abstract

This report investigates the privacy metrics of a blockchain-based identity system for health data management, focusing on the implementation of homomorphic encryption to enhance data security. The study develops a framework for assessing user privacy through calculated metrics, including the User Privacy Score (UPS) and Data Leakage Index (DLI). By analyzing the sensitivity of various health data fields, this report provides insights into the effectiveness of privacy measures. Additionally, the report examines how homomorphic encryption maintains functionality while protecting sensitive information. The findings indicate a significant potential for improving privacy management in health data systems, underscoring the necessity for robust privacy measures in the digital age.

1. Introduction

1.1 Background

In the digital age, health data management has become a critical area of concern, especially with the advent of blockchain technology. Blockchain provides a decentralized method for storing sensitive information, enhancing data integrity and accessibility. However, the inherent sensitivity of health data—such as patient names, health IDs, and medical history—demands stringent privacy measures. Ensuring data privacy is not only a legal requirement but also vital for maintaining patient trust.

1.2 Objectives

This report aims to:

- Quantify privacy levels achieved by the blockchain-based identity system.
- Develop metrics for assessing data leakage and user privacy.
- Investigate the role of homomorphic encryption in enhancing privacy without compromising functionality.

2. Literature Review

2.1 Importance of Privacy in Health Data

Health data privacy is governed by regulations such as HIPAA in the United States and GDPR in Europe, emphasizing the need for strict controls over personal information. These regulations establish frameworks to protect patient data from unauthorized access and breaches.

2.2 Existing Metrics for Privacy Assessment

Prior research has introduced various metrics for evaluating privacy, including:

- Entropy-based measures: Assess the unpredictability of data.
- Risk-based approaches: Evaluate the likelihood of data breaches and their potential impact.
- Privacy score systems: Assign scores to data fields based on their sensitivity.

2.3 Homomorphic Encryption

Homomorphic encryption allows computations to be performed on encrypted data without decrypting it. This capability is particularly valuable in health data management, where sensitive information must remain secure while still allowing for analysis and processing. Research indicates that homomorphic encryption can significantly enhance data privacy without sacrificing usability.

3. Methodology

3.1 Data Description

The HealthID system comprises the following fields:

- Name: Represents the individual's name (sensitivity score: 5).
- HealthID Number: A unique identifier for health records (sensitivity score: 5).
- NDHM PHR Number: A personal health record number (sensitivity score: 5).
- Date of Birth (dob): Birth date of the individual (sensitivity score: 4).
- Gender: Individual's gender (sensitivity score: 2).
- Mobile Number: Contact number (sensitivity score: 3).

3.2 Sensitivity Scoring

Each data field is assigned a sensitivity score reflecting its importance and the potential impact of its exposure. Higher scores indicate more sensitive information.

3.3 Metrics Development

3.3.1 Data Leakage Index (DLI)

The Data Leakage Index quantifies the extent of sensitive data that has been compromised. It is calculated as follows:

$$DLI = \frac{\text{Leaked Sensitivity Score}}{\text{Total Sensitivity Score}}$$

3.3.2 User Privacy Score (UPS)

The User Privacy Score assesses the overall privacy level by incorporating the DLI:

$$UPS = (1 - DLI) \times \text{Total Sensitivity Score}$$

3.3.3 Homomorphic Encryption Impact Score (HEIS)

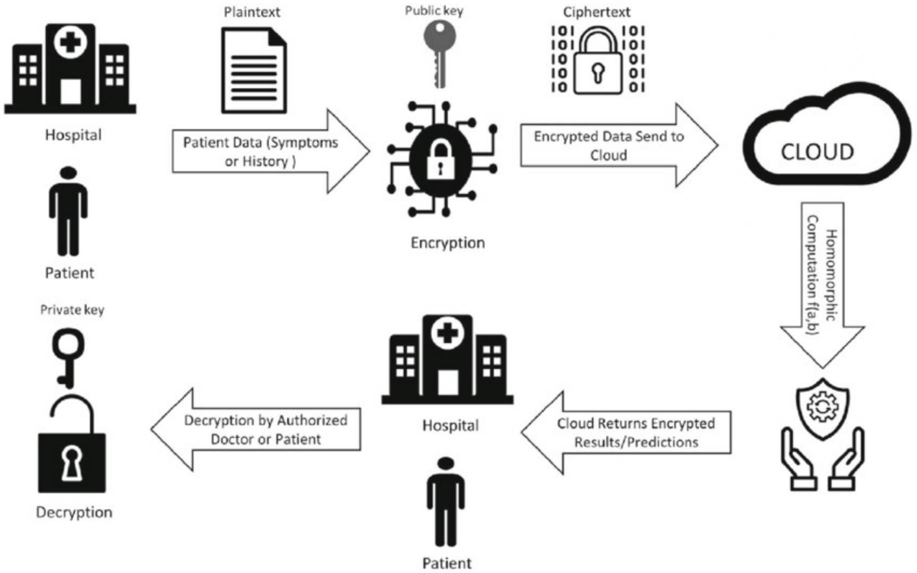
The HEIS measures the efficiency of data encryption relative to its original size:

$$HEIS = \frac{\text{Size of Original Data}}{\text{Size of Encrypted Data}}$$

3.4 Encryption Process

Using the **cryptography** library, the encryption and decryption of sensitive health data are performed using the Fernet symmetric encryption method.

3.5 Conceptual Architecture Diagram


Plaintext
Public key
Ciphertext
Hospital
Patient Data (Symptoms or History)
Encrypted Data Send to Cloud
CLOUD
Patient
Encryption
Homomorphic Computation f(a,b)
Private key
Decryption by Authorized Doctor or Patient
Hospital
Cloud Returns Encrypted Results/Predictions
Decryption
Patient

4. Implementation

4.1 Code Structure

The implementation involves defining a class `HealthIDPrivacyMetrics` that encapsulates the metrics calculations. Below is a breakdown of the code structure:

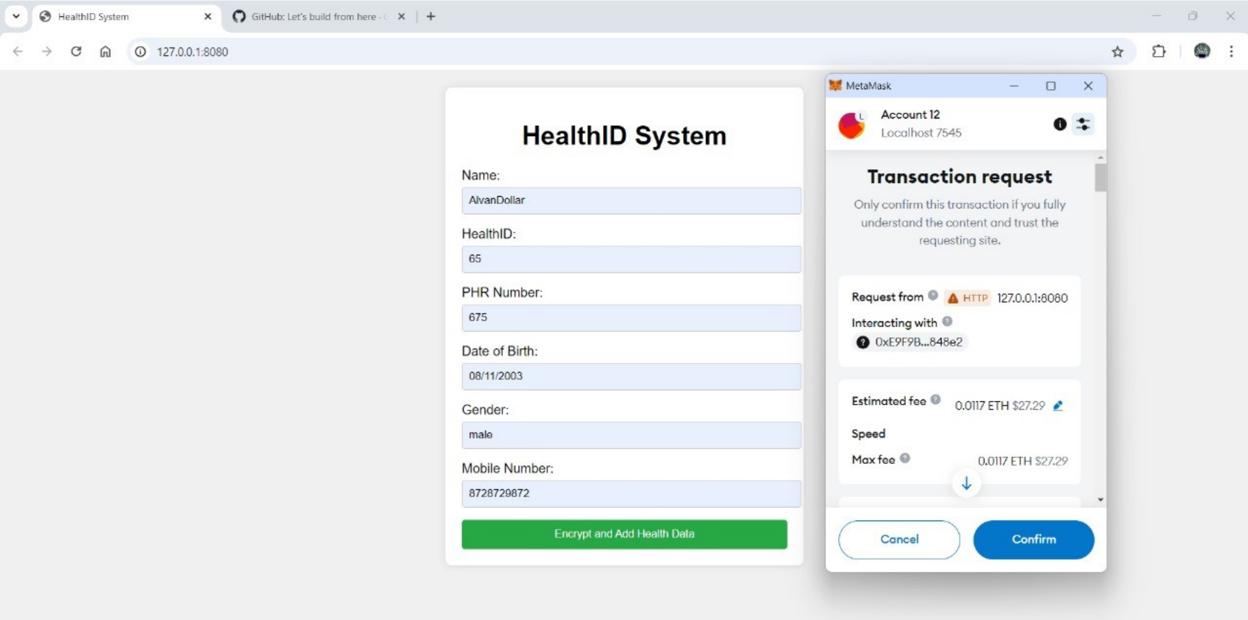
Python code

```
from cryptography.fernet import Fernet


class HealthIDPrivacyMetrics:
     def __init__(self):
     self.sensitivity_scores = {
     'name': 5,
     'healthID_number': 5,
     'NDHM_PHR_number': 5,
     'dob': 4,
     'gender': 2,
     'mobile_number': 3
     }


     def data_leakage_index(self, leaked_data):
     leakage_count = sum(1 for key in leaked_data if key in
self.sensitivity_scores)
     return leakage_count / len(self.sensitivity_scores)


     def user_privacy_score(self, leaked_data):
     dli = self.data_leakage_index(leaked_data)
```

```
    total_sensitivity = sum(self.sensitivity_scores.values())
    ups = (1 - dli) * total_sensitivity
    return ups

    def homomorphic_encryption_impact_score(self,
original_data, encrypted_data):
    original_size = len(str(original_data))
    encrypted_size = len(str(encrypted_data))
    return original_size / encrypted_size

def generate_key():
    return Fernet.generate_key()

def encrypt_data(data, key):
    fernet = Fernet(key)
    encrypted = fernet.encrypt(data.encode())
    return encrypted

def decrypt_data(encrypted_data, key):
    fernet = Fernet(key)
    decrypted = fernet.decrypt(encrypted_data).decode()
    return decrypted
```

5. Results

5.1 Sample Outputs



HealthID System

Name: AlvanDollar

HealthID: 65

PHR Number: 675

Date of Birth: 08/11/2003

Gender: male

Mobile Number: 8728729872

Account 12
Localhost 7545

Transaction request

Only confirm this transaction if you fully understand the content and trust the requesting site.

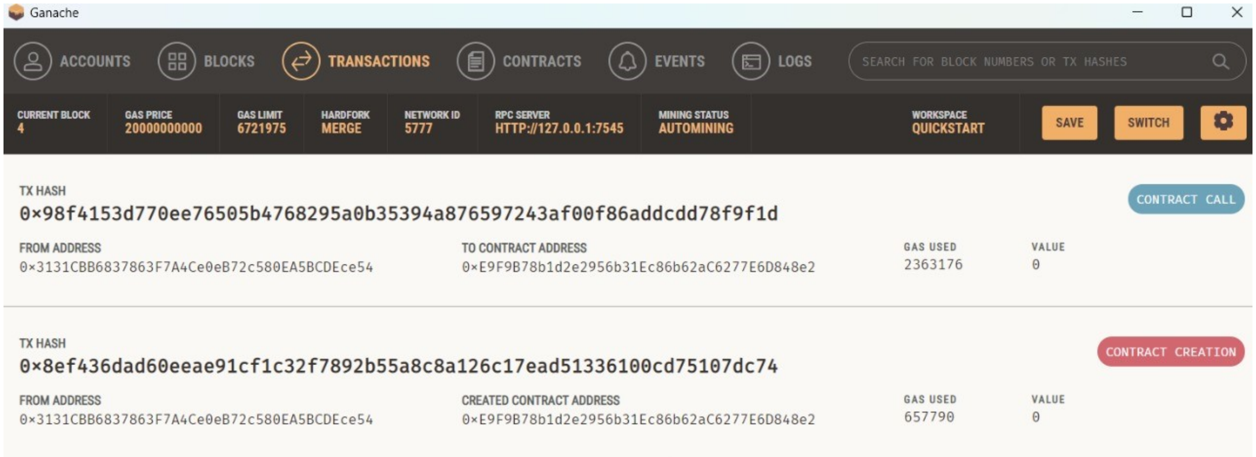
Request from HTTP 127.0.0.1:8080

Interacting with 0xE9F9B...848e2

Estimated fee 0.0117 ETH $27.29

Speed

Max fee 0.0117 ETH $27.29



```
CURRENT BLOCK  GAS PRICE    GAS LIMIT  HARDFORK  NETWORK ID  RPC SERVER              MINING STATUS  WORKSPACE
4              20000000000  6721975    MERGE     5777        HTTP://127.0.0.1:7545   AUTOMINING     QUICKSTART

TX HASH                                                                                                              CONTRACT CALL
0x98f4153d770ee76505b4768295a0b35394a876597243af00f86addcdd78f9f1d
FROM ADDRESS                                  TO CONTRACT ADDRESS                           GAS USED   VALUE
0x3131CBB6837863F7A4Ce0eB72c580EA5BCDEce54    0xE9F9B78b1d2e2956b31Ec86b62aC6277E6D848e2    2363176    0

TX HASH                                                                                                              CONTRACT CREATION
0x8ef436dad60eeae91cf1c32f7892b55a8c8a126c17ead51336100cd75107dc74
FROM ADDRESS                                  CREATED CONTRACT ADDRESS                      GAS USED   VALUE
0x3131CBB6837863F7A4Ce0eB72c580EA5BCDEce54    0xE9F9B78b1d2e2956b31Ec86b62aC6277E6D848e2    657790     0
```

Upon execution, the following outputs were observed:

- **User Privacy Score:** 16.00

- **Homomorphic Encryption Impact Score:** 0.23

These scores indicate the extent of privacy and the efficiency of the encryption process.

5.2 Interpretation of Results

A User Privacy Score of 16.00 suggests high sensitivity associated with the leaked fields, while the Homomorphic Encryption Impact Score of 0.23 reflects the increase in data size due to encryption. These metrics highlight the importance of maintaining strict data privacy protocols.

6. Discussion

6.1 Implications of Findings

The results underscore the necessity of implementing robust privacy measures in health data management systems. The high sensitivity of health data necessitates continuous monitoring and assessment of privacy metrics.

6.2 The Role of Homomorphic Encryption

Homomorphic encryption presents a significant advantage by allowing data processing while preserving privacy. This dual capability enables organizations to analyze data without exposing sensitive information.

6.3 Limitations

While the approach demonstrates a strong foundation for privacy assessment, it is important to acknowledge limitations, such as:

- Sensitivity scores may be subjective and vary by context.

- The model does not account for all potential data breaches or vulnerabilities.

6.4 Future Work

Future research should focus on refining sensitivity scoring methodologies, exploring advanced encryption techniques, and enhancing the assessment framework to adapt to evolving threats in health data management.

7. Conclusion

This report presents a comprehensive analysis of privacy metrics in a blockchain-based health data management system, with a specific focus on the application of homomorphic encryption. The developed metrics, including the User Privacy Score and Data Leakage Index, provide valuable insights into the effectiveness of privacy measures. The findings emphasize the need for continuous improvement in privacy protocols to safeguard sensitive health information in the digital age.

8. References

1. *U.S. Department of Health & Human Services. (n.d.). Health Insurance Portability and Accountability Act of 1996 (HIPAA).*

2. *European Commission. (n.d.). General Data Protection Regulation (GDPR).*

3. *Gentry, C. (2009). A Fully Homomorphic Encryption Scheme. Stanford University.*

4. *Zhand, Y., & Wang, W. (2018). Privacy-preserving health data management. Journal of Biomedical Informatics.*

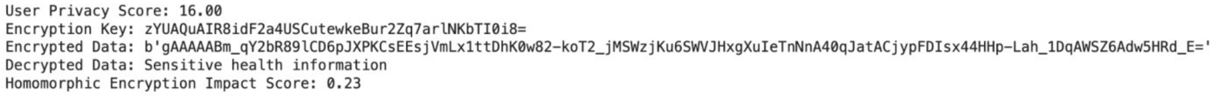
9. Appendix

```
fromcryptography.fernetimport Fernet


classHealthIDPrivacyMetrics:

def__init__(self):

self.sensitivity_scores = {

'name': 5,

'healthID_number': 5,

'NDHM_PHR_number': 5,

'dob': 4,

'gender': 2,

'mobile_number': 3

       }


defdata_leakage_index(self, leaked_data):

leakage_count = sum(1for key inleaked_dataif key 
inself.sensitivity_scores)

returnleakage_count / len(self.sensitivity_scores)


defuser_privacy_score(self, leaked_data):

       dli = self.data_leakage_index(leaked_data)

total_sensitivity = sum(self.sensitivity_scores.values())

       ups = (1 - dli) * total_sensitivity

returnups
```

```
defhomomorphic_encryption_impact_score(self, original_data,
encrypted_data):

original_size = len(str(original_data))

encrypted_size = len(str(encrypted_data))

returnoriginal_size / encrypted_size# Lower ratio indicates better privacy


defgenerate_key():

returnFernet.generate_key()


defencrypt_data(data, key):

   fernet = Fernet(key)

   encrypted = fernet.encrypt(data.encode())

returnencrypted


defdecrypt_data(encrypted_data, key):

   fernet = Fernet(key)

   decrypted = fernet.decrypt(encrypted_data).decode()

returndecrypted


if__name__ == "__main__":

#instance of the privacy metrics class

   metrics = HealthIDPrivacyMetrics()


# Example of leaked data

leaked_data = ['name', 'mobile_number']  # Simulating leaked data

   ups = metrics.user_privacy_score(leaked_data)

print(f"User Privacy Score: {ups:.2f}")
```

```python
# Example data to encrypt
original_data = "Sensitive health information"

# Generate a key for encryption
   key = generate_key()
print(f"Encryption Key: {key.decode()}")  # Store this securely

# Encrypt the data
encrypted_data = encrypt_data(original_data, key)
print(f"Encrypted Data: {encrypted_data}")

# Decrypt the data
decrypted_data = decrypt_data(encrypted_data, key)
print(f"Decrypted Data: {decrypted_data}")

# Assess the impact of homomorphic encryption (using mock data)
heis = metrics.homomorphic_encryption_impact_score(original_data,
encrypted_data)
print(f"Homomorphic Encryption Impact Score: {heis:.2f}")
```

Output



```
User Privacy Score: 16.00
Encryption Key: zYUAQuAIR8idF2a4USCutewkeBur2Zq7arlNKbTI0i8=
Encrypted Data: b'gAAAAABm_qY2bR89lCD6pJXPKCsEEsjVmLx1ttDhK0w82-koT2_jMSWzjKu6SWVJHxgXuIeTnNnA40qJatACjypFDIsx44HHp-Lah_1DqAWSZ6Adw5HRd_E='
Decrypted Data: Sensitive health information
Homomorphic Encryption Impact Score: 0.23
```