National College of Ireland

Homomorphic Encryption as a Counter Measure for Data Breach and Insider Threat

MSc Research Project
Programme Name

Anurodhan Pradhan
Student ID: X22134638

School of Computing
National College of Ireland

Supervisor: Eugene Mclaughlin

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name: …….Anurodhan Pradhan……

Student ID: ……X22134638…….……

Programme: …MSc in Cybersecurity…… **Year:** ……2023-24..

Module: ……MSc Research Project…….……

Supervisor: ……Eugene Mclaughlin…….……

Submission Due Date: ……14 December 2023…….……

Project Title: Homomorphic Encryption as a Counter Measure for Data Breach and Insider Threat

Word Count: ……6161…… **Page Count**……20…….……..

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: ……Anurodhan Pradhan……

Date: ……14 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):	

Homomorphic Encryption as a Counter Measure for Data Breach and Insider Threat

Anurodhan Pradhan
X22134638

Abstract

In the digital era, where data breaches and insider threats are becoming common, securing sensitive data has become crucial. This research paper dives into Homomorphic Encryption (HE) as an innovative solution to these cybersecurity challenges, comparing it with traditional encryption methods. The motivation behind this study is driven by the need for encryption techniques that allow data processing while maintaining absolute confidentiality, even after the data gets exposed. This study assesses HE's ability to ensure data privacy during computing and resist potential quantum computing threats. This research compares HE with traditional encryption techniques, emphasising HE's unique potential to carry out calculations on encrypted data and providing a better way of maintaining data integrity and confidentiality in complex processing scenarios. This research plays a significant role in encountering data privacy and security threats. The advanced level of security enables the computation of encrypted data. However, a substantial amount of computation overhead and complexity will be there. Despite these challenges, the potential of HE to safeguard data against insider threats, data breaches and future quantum attacks makes this research important to the end users, which includes organisations and individuals. In this report, we presented an algorithm to perform HE on strings, numbers, and special characters. The implementation and details of the algorithm and results are shown.

1 Introduction

In this decade of digital technology, there has been a significant change in the data security in managing the data. Primarily, it focuses on the advanced measures for protecting it inside-out. The significant change has been driven by massive technological advancements, which have transformed the whole data structure, including data storage, access, and protection mechanisms (Patel, 2023). The technological revolution has led to exponential growth in the amount of data generated, stored, and processed online. From personal information to critical organisational data. Everything is stored digitally nowadays—the only way to get targeted by cybercriminals (ACAR, et al., 2018). The increase in digitalisation has also elevated the risks associated with data breaches, where unauthorised access to the data can lead to severe consequences, such as identity theft, which can end up causing substantial financial loss and reputation damage.

We use the data by accessing it and utilising it with the help of cloud computing and in-house servers. The data is no longer secure. It is frequently accessed and processed over various networks, sometimes over international boundaries. This widespread distribution of data poses a significant challenge in ensuring its security as the traditional structure and

defence mechanism to secure the data is no longer adequate as the technology is evolving with that cybercriminal as well.

Advanced encryption methods are needed to overcome this challenge. Traditional encryption techniques are effective in a specific context but have some limitations in the new digital technology or digital environment. It often requires data to be decrypted for processing, which leads to a potential vulnerability exposure. Additionally, the increase in computation power available to cyber criminals has made it specific to breaking down the older encryption methods with the resources. According to the sources, 34% of businesses globally suffer the consequences of insider threats yearly, and 6.41 million of data was breached in the first quarter of 2023 (Sectara, 2023) (Petrosyan, 2023).

As a result, the technology has switched to creating more robust and flexible encryption solutions that can protect data not only while it is in rest or in transit but also while the data is computing. Here is where HE comes into play, providing a novel way to safeguard data while it is in computation. HE encryption ensures the confidentiality and integrity of the data throughout its lifecycle by enabling the computations on encrypted data without needing to decrypt it first (Alaya, et al., 2023). This is how HE will prevent the insiders from accessing the data. Suppose the data is exposed in an incident. In that case, the data is secure due to the latest form of encryption, which can only be decrypted by a corresponding secret key. This research will address the current security gap in the digital era (TEK˙IN, 2023).

1.1 Question and Answer

Table 1: Research question and answer

Q1. What are the two primary security challenges faced by organisations in the digital age?	HE addresses both data breaches and insider threats by allowing secure computations on encrypted data.
Q2. How do data breaches and insider threat typically occur in organisations?	HE mitigates data breaches and insider threat by keeping data encrypted even during processing, eliminating the need for decryption in vulnerable environments.
Q3. What are the limitations of traditional encryption methods in preventing data breaches and insider threats?	HE overcomes the limitations of traditional encryption by encrypting data throughout its lifecycle, reducing the risk of exposure during processing.
Q4. What role do vulnerabilities in traditional encryption play in facilitating insider threats and data breach?	HE reduces the window of opportunity for insider threats and data breaches by ensuring data remains encrypted even during computing, unlike traditional methods.

1.2 Structure of the report

The report begins with abstract an in section 1 an introduction detailing the background, problem statement, solution strategy is described , followed by literature review on data breaches, insider threats, and HE and its types in section 2. A research methodology describing the research design and analysis techniques in section 3, a design specification explaining the technology, techniques, and algorithms in section 4, an Implementation discussing the use of tools and HE solutions in section 5, an evaluation assessing implementation across various data types in section 6. A discussion and comparative study on HE and other tradition algorithms in section 7. A conclusion and future work highlighting key findings and potential future research directions in section 8 and at last references in section 9.

2 Related Work

2.1 Data breach

A data breach is an event when private, sensitive, or protected data is accessed or revealed without authority. It frequently results from security flaws like hacking, insider threats, or human error, which puts the people and organisations involved at risk of identity theft, financial loss, and reputational harm. This paper focuses on the creation of a database for data breach incidents, drawing from various public resources to analyze and understand trends in data breaches, particularly from 2018 to 2019. It addresses the challenges and limitations involved in building such a database, underscoring the importance of encryption and additional security measures for data protection. The research aims to enhance data security and privacy by providing insights into identity theft, financial losses, and reputational harm caused by data breaches due to hacking, insider threats, or human error. The findings offer valuable recommendations for improving organisational strategies and standard operating procedures to prevent future data breaches. (NETO, et al., 2021). At the end of the attack, the cybercriminals want the victim's data. To enhance data security, this research will fill the technical gap with the help of HE as a countermeasure against data breaches and insider threats. We will use HE to improve security. However, something happens, and the hackers manage to get access to the data, but they cannot read or decrypt it. Thus, the data is secured, reducing the risk of data breaches.

2.2 Insider Threat

An insider threat is a security risk from within the organisation and usually involves a person given authorised access, such as employees, business partners and contractors. This paper presents a detailed classification of insider threats into seven categories, including IT sabotage, fraud, intellectual property theft, social engineering, unintentional incidents, cloud computing, and national security threats. It explores how these threats impact organisational security objectives and are driven by human behavior. While offering insights into both technical and non-technical methods for mitigating these risks, the paper notes a lack of supporting data and practical case studies. It emphasizes the need for deeper analysis of preventive measures and the adaptation of these generic strategies to specific organisational contexts, particularly in the face of growing cloud computing risks. At the end of the attack, the cybercriminals want the victim's data. To enhance data security, this research will fill

the technical gap with the help of HE as a countermeasure against insider threats. We will use HE to improve security. However, something happens, and the hackers manage to get access to the data, but they cannot read or decrypt it. Thus, the data is secured, reducing the risk of data breaches (Elmrabit, et al., 2015).

2.3 Homomorphic Encryption and its types

This paper (Ogburn, et al., 2013) delves into HE, a method that enables computations on encrypted data, producing results that align with operations on plaintext, thereby maintaining data privacy during processing. It examines the theoretical aspects and different types of HE, including fully, somewhat, and partially homomorphic encryption, focusing on its application in cloud computing for secure data processing. The paper includes a proof-of-concept to demonstrate practical implementation but tends to be narrow in scope, mainly addressing specific applications. It highlights the need for a deeper exploration of real-world challenges, such as efficiency and computational demands, to enhance the practical applicability of HE in various industries. These types differ in their capabilities and limitations:

2.3.1 Fully HE (FHE): Fully Homomorphic Encryption (FHE) has a unique ability to perform a variety of calculations, including multiplication and addition, directly on encrypted data, thereby enabling computations of any complexity while ensuring unparalleled data privacy and security. However, the significant computational complexity of FHE limits its practicality, particularly in real-time processing scenarios where efficiency and speed are crucial, making it less effective compared to other forms of HE.

2.3.2 Somewhat HE (SWHE): Somewhat Homomorphic Encryption (SWHE) enables limited addition and multiplication operations on encrypted data, suitable for tasks with known and set computational complexity (Yang, 2012). SWHE offers a tailored approach to processing encrypted data, balancing operational practicality and security. Its efficiency advantage over FHE makes it a more viable option for practical applications with limited computing needs.

2.3.3 Partial HE (PHE): Partially Homomorphic Encryption (PHE) excels in supporting only one operation, either addition or multiplication, on encrypted data, making it ideal for specific applications that require a single type of computation. Its focused operational ability ensures efficiency and practicality in specialized settings, especially where speed and simplicity are essential and computational demands are limited to a single, well-defined operation.

Every one of these encryption techniques has benefits and drawbacks of its own. The most computational flexibility provided by FHE but at the expense of computational performance. While more efficient than FHE, SWHE maintains a balance by providing more computational choices than PHE. PHE has the lowest efficiency and is best suited for applications that need to perform single, repeating tasks despite its restricted capability. To enhance data security, this research will fill the technical gap with the help of HE as a countermeasure against insider threats. We will use HE to improve security. However, something happens, and the hackers manage to get access to the data, but they cannot read or decrypt it. Thus, the data is secured, reducing the risk of data breaches. In the upcoming paper, we will discuss the implementation of HE.

2.4 Implementing Homomorphic Encryption

This paper (S, et al., 2022) explores the concept of HE in cryptography, highlighting its ability to perform computations on encrypted data without decryption, thus enhancing data security, particularly in cloud storage and big data contexts. It discusses the evolution of HE, notably through FHE by Gentry, which allowed for more complex computations compared to previous systems like RSA. The paper proposes an innovative method combining HE with Advanced Encryption Standard (AES) in Cipher Block Chaining mode, addressing significant data security concerns. However, it acknowledges the complexity and computational demands of this approach, particularly with AES. It suggests that more analysis is needed to fully understand the security vulnerabilities of this integrated system. Despite these challenges, the proposed system is a step forward in securing string operations and data types with minimal computational overhead. To enhance data security, this research will fill the technical gap with the help of HE as a countermeasure against insider threats. We will use HE to improve security. However, something happens, and the hackers manage to get access to the data, but they cannot read or decrypt the data. Thus, the data is secured, reducing the risk of data breaches.

This paper (Vemula, et al., 2023) delves into the advancements in HE, particularly focusing on FHE for secure cloud-based data storage. It highlights the development of FHE algorithms, which allow unlimited arithmetic operations on encrypted data, emphasizing their practical application in scenarios requiring secure cloud computation. The paper notably explores Craig Gentry's groundbreaking work in FHE, underscoring its capability for arbitrary computations on encrypted data without compromising confidentiality. This is crucial for sectors like finance and healthcare requiring high data privacy. The paper also discusses FHE's computational challenges, including Gentry's bootstrapping method, and points out the need for more practical implementations and prototypes to address its limitations, suggesting avenues for future research. It also highlights the necessity to compare FHE with other encryption methods to evaluate its efficiency and effectiveness, acknowledging differences in capabilities and limitations. To enhance data security, this research will fill the technical gap with the help of HE as a countermeasure against insider threats. We will use HE to improve security. However, something happens, and the hackers manage to get access to the data, but they cannot read or decrypt it. Thus, the data is secured, reducing the risk of data breaches.

2.5 Literature Summary and gap

From the literature survey above, despite advancements in data security and encryption, current approaches face challenges like limited scope, computational inefficiencies, and a lack of comprehensive security evaluations. This underscores the need for further research into advanced and efficient encryption methods, particularly improved versions of HE, to address these limitations. Our research aims to bridge this gap by proposing HE as a robust solution against evolving cybersecurity threats, focusing on enhancing data security against insider threats and data breaches across various applications.

2.6 Inferences Drawn

Table 2: Inferences drawn from literature survey

Authors and Paper name	Year of Publication	Significance
[1]N. N. NETO, S. MADNICK, A. M. G. D. PAULA and N. M. BORGES, "Developing a Global Data Breach Database and the Challenges Encountered"	2021	Creating a database to analyse patterns in data breaches which emphasizing the need of encryption and security measures in databases.
[2] N. Elmrabit, S.-H. Yang and L. Yang, "Insider Threats in Information Security"	2015	Evaluates techniques for mitigating insider threats and categorise them to understand and control these risk, technology controls and human behaviour analysis are integrated.
[3]M. Ogburn, C. Turner and P. Dahal, "Homomorphic Encryption"	2013	Describes the three different forms of HE and offers theoretical explanations of their advantages and disadvantages for processing data securely.
[4] R. S, V. B, A. B. Mehta and P. B. Honnavalli, "Homomorphic Encryption Approach for String"	2022	The integration of HE with the Advanced encryption Standard(AES) in cipher block chaining mode is examined in this study, with a specific focus on safe string operations in big data and cloud computing.
[5] S. Vemula, R. M. R. Kovvur and D. Marneni, "Algorithms for Implementing Repeated Homomorphic Operations on Restricted Data Type,"	2023	It particularly focuses on its use in cloud-based data storage, along with craig Gentry's Fully HE approaches, which allows arbitrary computations on encrypted data without the need for decryption.

In table 2 the significance of each paper has been described.

3 Research Methodology

This research aims to demonstrate the implementation of HE of strings, numbers and special character using Python; it is a powerful form of encryption that uniquely allows computation to be performed directly on encrypted data without compromising its confidentiality. This feature of HE differentiates it from traditional encryption techniques and creates novel prospects for safe data processing in Complex environments.

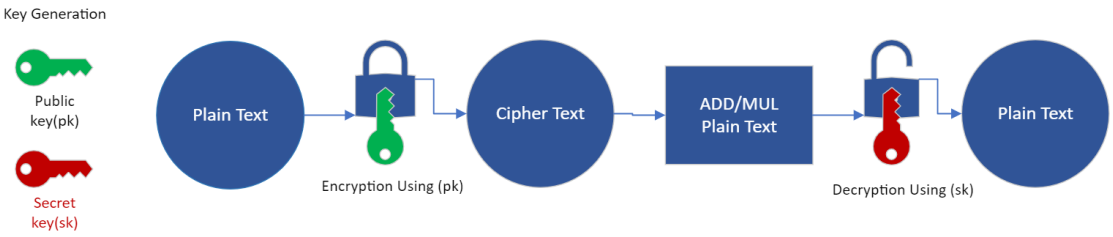

Figure 1: Workflow of the proposed system

3.1 Key Generation

Key generation **keygen()** is a crucial process in cryptography (Abdalrdha, et al., 2019). It creates a pair of cryptographic keys as the public and Secret keys. These keys are used for securing the communication and data. The key generation process usually involves selecting a large random number and performing mathematical operations to generate the key pair. This is also depending on the cryptographic algorithm that has been used.
The purpose of key generation is to generate pair keys using **keygen(),** which will be used for encryption and decryption of data.

3.2 Public Key

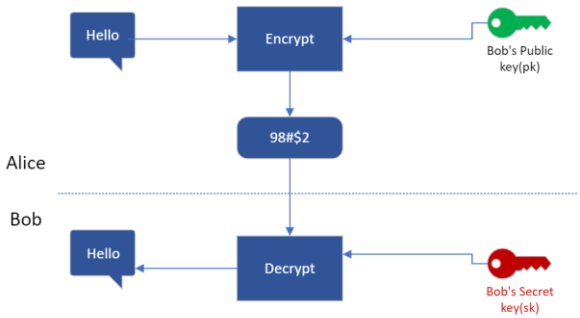

Figure 2: Public key

This is a publicly shared key **pk()** used to encrypt data. In most cryptographic systems, anything encrypted using a public key can only be decrypted by a suitable Secret(private) key

(Hellman, 1978). A public key is a fundamental cryptography component used for secure data transmission.
The purpose of public key **pk()** is to encrypt the data.

3.3 Secret Key

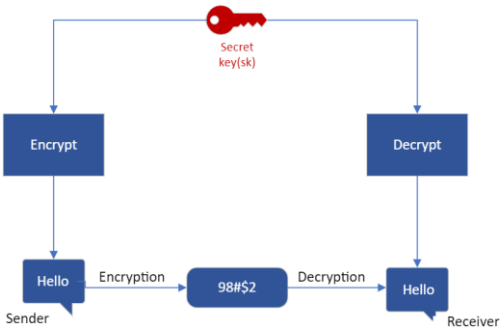

Figure 3:Secret key

The secret key **sk()** is kept confidential, and it is used to decrypt the data that has been encrypted with the public key. It is also used in some systems as a digital sign document, ensuring the authenticity and integrity of the data. A secret key is a crucial cryptography component, forming one-half of a cryptographic key pair.
The secret key **sk()** is to decrypt the encrypted data.

3.4 Encryption

It is a process in cryptography where data is referred to as plaintext and is converted into a coded form known as ciphertext. This transformation is done to prevent unauthorized access or reading of the data. The primary purpose of encryption is to protect the confidentiality of digital information stored on the computer or transmitted via the internet or another network (S.Suguna, et al., 2016).

3.4.1 Types of encryptions:

Symmetric encryption: Data encryption and decryption with a single key are the characteristics of symmetric encryption. The key, known as the secret key, needs to be shared between the parties to communicate. Because of its efficiency and speed, it is ideal for encrypting considerable amounts of data. Common algorithms such as Triple DES, Data Encryption Standard (DES), and Advance Encryption Standard (AES) are used.

Asymmetric encryption: This is also known as public key encryption, in which two keys are used: a public key for encrypting the data and a private or secret key to decrypt the data. The private key needs to be kept confidential. Only the matching private key can decrypt the encrypted data. It is specially used for establishing secure communication over insecure channels like the internet and for the digital signature. Algorithms like Elliptic curve cryptography (ECC) and Rivest-Shamir-Adleman(RSA) are used in this encryption.

The purpose of encryption is to encrypt the data using the public key. Here, the **encrypt()** takes the plain text and the public key as input and encrypts the data as cipher text.

3.5 Decryption

It is a process of converting encrypted data(cipher text) back into its original text(plain text), making it understandable. It is also the reverse operation of encryption in the field of cryptography. The primary purpose of the **decrypt()** is to convert the unreadable, encrypted data back into its original, readable form, with the secret key allowing authorized users to access and interpret the information.

3.6 Operation

The core component of this project, and its primary objective, revolves around executing arithmetic operations on encrypted data using HE. This is accomplished through two functions: **'add_plain'** and **'mul_plain'**. The **'add_plain'** function performs addition between ciphertext and plaintext, giving new encrypted data without decrypting the original ciphertext, thereby maintaining data security and confidentiality. Similarly, the **'mul_plain'** function multiplies plaintext with ciphertext, producing a new encrypted result, ensuring the data remains encrypted throughout the process.

3.7 Evaluation

The effectiveness of HE in this project is evaluated through various measurements using a Python framework.
Functional accuracy: is assessed by controlled tests on encryption, processing, and decryption, ensuring system integrity and reliability.
Performance metrics: focus on processing speed and resource utilization, identifying bottlenecks and scalability.
Security analysis: tests the system's robustness against cryptographic attacks and assesses algorithm strength for data privacy and protection.
Usability and flexibility: evaluation looks at the ease of integration and adaptation in various applications, enhancing developer accessibility.
Finally, a comparative analysis positions HE against traditional encryption methods, highlighting its unique advantages and limitations in performance, security, and usability.

4 Design Specification

Figure 4: Flow chart

We have illustrated the process of proposed methodology for the implementation of the HE in Figure 4 which is followed by:

4.1 Key Generation

The key generation process in HE starts with creating a secret key **‘sk’** as a binary polynomial, where coefficients are either 0 or 1, and its degree is determined by the parameter **'size'**. This key is crucial for decryption. Next, a polynomial **‘a’** with uniformly random coefficients is created, each chosen from a uniform distribution within a specified modulo space defined by **‘modulus’**. Following this, a noise polynomial **‘e’** is generated, with coefficients from a normal distribution, essential for the scheme's security. The polynomial **‘b’** is computed by multiplying ‘a’ with the negative of **‘sk’** and adding the noise polynomial **‘e’** denoted as ‘**polymul(-a, sk, modulus, poly_mod)’** and **‘polyadd(polymul(-a, sk, modulus, poly_mod), -e, modulus, poly_mod)’** respectively, followed by modular reduction and division by the polynomial modulus **‘poly_mod’**. The public key **‘pk’** is formed as the tuple **‘(b, a)’**, facilitating the encryption process. This method ensures the security of the encryption process by involving the secret key in the formation of the public key without direct usage in encryption, leveraging polynomial arithmetic under modular constraints.

4.2 Encrypt String

The 'Encrypt String' process in HE involves converting each character of an input string into its corresponding ASCII (American Standard Code for Information Interchange) value, a standard numerical representation used in electronic communication. For instance, **'A'** and **'B'** are represented as 65 and 66, respectively. This conversion is essential since the encryption operates on numerical values. Each ASCII value is then encrypted individually: it is encoded as a polynomial with the integer as its constant term and all other coefficients set to zero. This polynomial undergoes scaling, aligning it with the ciphertext space, and is fortified with added noise for enhanced security, making it resilient against various cryptographic attacks.

The encryption process includes polynomial arithmetic using the public key, where the plaintext polynomial is combined with the polynomials from the public key and additional noise, culminating in a tuple of polynomials that represent the encrypted ASCII value. The outcome of the **'encrypt string'** function is an array of such tuples, each signifying the encrypted form of a character from the original string (Gangula, 2022). This array can be securely stored or transmitted, ensuring that only individuals with the appropriate secret key can decrypt it back to the original string. In essence, the **'encrypt_string'** function transforms each character into an ASCII value, encrypts it using polynomial based HE, and outputs an array of tuples representing the encrypted string.

4.3 Encryption Process

Encryption in HE specifically focuses on encrypting ASCII values of characters as integers. The process begins with encoding an integer into a polynomial **'m'**, setting it as the constant term while other coefficients are zero. For example, the integer 65 is encoded into the polynomial **'m'** as $65 + 0x + 0x^2 + ...$ up to the required size. This polynomial is then scaled by a delta factor, derived from the ciphertext modulus **'q'** and plaintext modulus **'t'**, aligning the plaintext with the ciphertext space, crucial for HE's functionality. Noise polynomials **'e1'** and **'e2'**, generated from a normal distribution, along with a random binary polynomial **'u'**, add cryptographic security and randomness to each encryption. The ciphertext components

'ct0' and **'ct1'** are calculated using polynomial arithmetic with the public key, under the modular arithmetic defined by **'q'**. The resulting tuple **'(ct0, ct1)'** represents the encrypted integer, encapsulating encrypted data in a manner that permits computations without decryption, thus fulfilling the homomorphic property.

4.4 Decrypt String

The **'decrypt_string'** function decrypts an encrypted string by processing an array of ciphertext tuples, each representing an encrypted ASCII value of a character. The decryption involves two main steps: first, each ciphertext tuple undergoes polynomial arithmetic using the secret key, reversing the encryption process. This includes polynomial multiplication, addition, modular reduction, and scaling down to map the ciphertext back to plaintext space, thus retrieving the original ASCII values. Second, these decrypted ASCII values are converted back to their corresponding characters based on the ASCII standard (e.g., an integer 65 is converted to **'A'**). The function iteratively decrypts each tuple in the ciphertext array, reconstructing the original string by converting the decrypted integers back to characters, effectively reversing the encryption process.

4.5 Decryption Process

The 'decrypt' function is key in reversing the encryption process in HE. It starts by performing polynomial arithmetic with the secret key and ciphertext, where 'ct1' is multiplied by the secret key, and 'ct0' is added, all under modular constraints. The resulting polynomial is then scaled down to align the values from ciphertext to plaintext space, followed by rounding off the coefficients to nearest integers to counteract fractional values introduced during encryption. The final step involves extracting the first coefficient of the scaled polynomial, representing the decrypted integer value. This process effectively retrieves the original plaintext integers through a series of strategic polynomial operations and adjustments.

4.6 Algorithm

Step 1: Key Generation (keygen)

- **Inputs: size** (polynomial size), **modulus**, **poly_mod** (polynomial modulus).
- Generate **sk** using **gen_binary_poly(size)**.
- Generate **a** using **gen_uniform_poly(size, modulus)**.
- Generate **e** using **gen_normal_poly(size)**.
- Calculate **b** using **polyadd(polymul(-a, sk, modulus, poly_mod), -e, modulus, poly_mod)**.
- **Outputs:** Public key (**pk**) as **(b, a)**, secret key (**sk**).

Step 2: Encrypt Integer (encrypt)

- **Inputs:** Public key (**pk**), integer (**pt**), **size**, **q** (ciphertext modulus), **t** (plaintext modulus), **poly_mod**.
- Encode **pt** into a polynomial **m**.
- Scale **m** to **scaled_m**.

- Generate **e1**, **e2**, and **u** using **gen_normal_poly(size)** and **gen_binary_poly(size)**.
- Calculate **ct0** and **ct1** using polynomial operations with **pk** and **u**.
- **Output:** Ciphertext as **(ct0, ct1)**.

Step 3: Encrypt String (encrypt_string)

- **Inputs:** Public key (**pk**), plaintext string.
- For each character:
  - Convert to ASCII value.
  - Encrypt using **encrypt** function.
- **Output:** Array of ciphertext tuples.

Step 4: Decrypt Integer (decrypt)

- **Inputs:** Secret key (**sk**), ciphertext tuple (**ct**), **size**, **q**, **t**, **poly_mod**.
- Compute the scaled polynomial from **ct** and **sk**.
- Scale down and round off the result.
- Extract the first coefficient as the decrypted integer.
- **Output:** Decrypted integer (ASCII value).

Step 5: Decrypt String (decrypt_string)

- **Inputs:** Secret key (**sk**), array of ciphertext tuples.
- For each tuple:
  - Decrypt using **decrypt** function.
  - Convert decrypted integer to character.
- **Output:** Decrypted string.

This pseudocode represents the key processes in HE algorithm, including key generation, encryption and decryption of both integers and strings. Each function is outlined with its inputs, operations, and outputs, providing a structured and clear representation of the algorithm's workflow.

5 Implementation

In this section, the whole implementation of HE, along with the result, tools used, and the libraries used, will be discussed. The goal was to implement HE in strings, which is under development in cryptography. Here, in this research, we will show the implementation of HE in Strings.

5.1 Tools and language used:

Python is chosen for its extensive libraries and tools, was utilized to implement HE in strings. The project, aimed at simplicity, effectiveness, and flexibility, did not use inbuilt HE libraries due to maintenance and resource demands.

Figure 5: NumPy

NumPy is chosen for its compatibility across systems and specialization in numerical computing, it offers efficient polynomial arithmetic operations crucial for HE. Its advantages include smooth system-wide code execution and fast computation capabilities.

5.2 Final Configuration of the solution:

This project successfully implements HE on strings, a crucial advancement as HE allows for computation on encrypted data without exposing it. With data increasingly stored digitally, traditional encryption methods are vulnerable to cybercriminals with advanced computing resources. HE stands out as a robust encryption technique, offering a simple, effective, and resource-efficient solution. This implementation, utilizing SWHE, marks a significant step in enhancing data security in the digital age.

This project focuses on implementing HE for string operations, specifically addition and multiplication, outputting results as encrypted numbers. These can be securely stored in databases and updated or retrieved using a secret key, effectively mitigating insider threats and data breaches. The encryption is robust enough to resist decryption even with advanced computing resources, requiring a quantum computer to break. HE facilitates secure data migration to the cloud, a capability lacking in traditional encryption methods. The use of NumPy enhances performance, making HE a crucial tool in data security across various digital storage technologies. Future work involves integrating this implementation into databases and data storage technologies.

6 Evaluation

In this section, we will critically assess the results and the findings of the implementation, focusing on the practical and theoretical aspects of the findings. We have implemented SWHE on strings. The arithmetic operations that are being performed are addition and multiplication. The purpose of the research is to assess the effectiveness of HE in data security against insider threats and data breaches. The evaluation is based on various parameters, which include data security, computational efficiency, and practical feasibility. Here are some of the case studies regarding the evaluation:

6.1 Case Study 1: Testing in Integers

```
Original String: 12
Encrypted String [(array([23547, 23294, 11692,  6502,  8851,  8773, 26589,  3033, 14950,
       19555, 30523,  9074,  6877, 29373, 22870,  8704], dtype=int64), array([17133, 15926, 18796,  4012, 23626, 19439,  6211, 24367, 29019,
        7047,  9023, 13736, 18232,  2193, 16744, 21270], dtype=int64)), (array([28953,  7950, 22473, 27110,   861,  4968,  1051, 28653, 22014,
        1871,  3984, 29397, 29787, 25977,  2945,  4457], dtype=int64), array([ 2495, 30378,   368, 24382, 21592, 23898, 19359, 23010,  9962,
       12601, 11763, 26963, 14921, 13615, 14428, 10762], dtype=int64))]
Decrypted String: 12
```

Figure 6: Output for integers

In figure 6 it shows the output of an integer as input which has been encrypted and being displayed in tuples as ciphertext and after the encryption the decryption is also shown in the output which is similar as input which defines the success of implementation of HE in Integers.

6.2 Case Study 2: Testing in special characters

```
Original String: *&$
Encrypted String [(array([ 4423, 12456,  1666, 21081, 16415, 21833, 12453, 29338, 22963,
       24654, 19534,  3411,  8042, 14224,  5177, 23022], dtype=int64), array([ 4810,  9120, 27135,  1648, 19685, 18048,  5347, 12562, 23864,
        9232,  5116, 29768,  1067, 13056, 14042,  1785], dtype=int64)), (array([ 4429, 13162,  4206, 19948, 32006,  8414, 25503, 15521, 27456,
       25487, 12754,  6965, 28890, 25641,  9475, 26220], dtype=int64), array([23036,  5145, 30019, 12219, 10859,  8614,  2976, 32331, 17093,
       18867,  1732, 18467,   270,  3790,  2582, 10615], dtype=int64)), (array([18478, 17180, 10358,  2487,  3572,  5916, 26504, 24524, 15489,
        4516, 18349, 24115,  5011, 17808, 18007, 32546], dtype=int64), array([19101, 11686, 23417, 12838,  7854,  4280,    30,  9534, 21162,
       11470, 23548, 17642, 17163, 15144, 27380, 18590], dtype=int64))]
Decrypted String: *&$
```

Figure 7: Output for special characters

In figure 7 it shows the output of symbols as input which has been encrypted and being displayed in tuples as ciphertext and after the encryption the decryption is also shown in the output which is similar as input which defines the success of implementation of HE in symbols.

6.3 Case Study 3: Testing in strings

```
Original String: Test
Encrypted String [(array([17755, 26601,  9216, 22501, 16411,  1147, 27617, 27528, 10761,
        2437, 31737,  9129,  1384, 29946, 14479, 11141], dtype=int64), array([15861,    70,  8883,  6649, 16885,   406, 21500,  5231, 26577,
       31626,  2544,  7281, 12515, 13728, 10399, 31798], dtype=int64)), (array([26796, 12912, 16688, 23364, 22637, 10117,  4047, 21115,   458,
        1357,  9160, 12324,  8959, 30579, 11186, 32171], dtype=int64), array([32156, 26235, 28030, 23432,  3663, 15041,  9745, 16067,  2616,
        7912, 10275, 25150,  8926,  7255,  1421,  4769], dtype=int64)), (array([ 4875,  9173, 23733, 14429, 26656,  1935, 30237, 12184, 11296,
       11878, 11594, 22461,  7521, 11307, 11966,  4094], dtype=int64), array([22088, 15535, 10171, 28153,  4718,  9061,  6884, 10374,  9614,
        3787, 22833, 21921,  1404,  2955,  8286, 26504], dtype=int64)), (array([10464,   556,  4254,  1408, 15633, 17472, 27004, 17310,  9622,
       30770,  6263,  6248, 23626,   473, 10132, 10289], dtype=int64), array([ 2248, 28887, 26067, 25843, 29283, 28118, 30270,  8545,  4920,
       12166,  9590, 18716, 26987,  3648,   617,  9825], dtype=int64))]
Decrypted String: Test
```

Figure 8:Output for strings

In figure 8 it shows the output of strings as input which has been encrypted and being displayed in tuples as ciphertext and after the encryption the decryption is also shown in the output which is similar as input which defines the success of implementation of HE in strings.

6.4 Case Study 4: Testing the combination of integers, special characters and strings

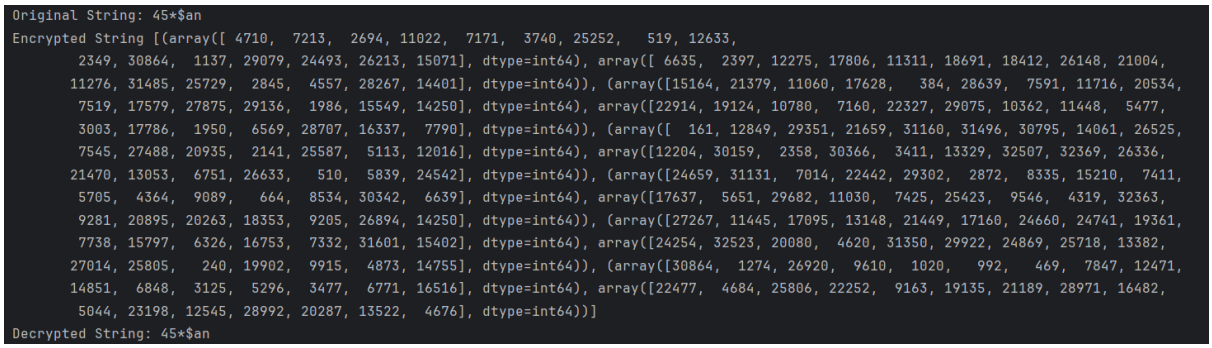

Figure 9:Output for combination of integer, special character and strings

In figure 9 it shows the output of combined input which has been encrypted and being displayed in tuples as ciphertext and after the encryption the decryption is also shown in the output which is similar as input which defines the success of implementation of HE.

7 Discussion

The key finding of this project is to maintain the data integrity and the confidentiality of the data using HE. This study shows that HE can be implemented on strings and in special characters and integers. This highlights flexibility in data security. With the help of this study, HE could be implemented in the database to perform certain operations on data. We have to run SQL queries, which include a mixer of special characters, integers and strings, which make HE advantageous over other encryption techniques. The implementation is easy and efficient using a standard and lightweight library. This improves the accessibility to HE but also makes it useful for devices with low processing power, giving more advantages over other traditional techniques that use huge libraries, which require high computer power. From an academic perspective, this research significantly contributes to cryptography and data security by implementing HE in real-world situations, especially cloud computing. A significant weakness in traditional encryption techniques has been addressed by HE to maintain the data encrypted even in processing. This report also emphasises the strategic significance of HE in preventing insider threats and data breaches. HE offers an essential layer of security in situations like data breaches and insider threats by ensuring that the compromised data stays safe and unreadable without the required secret key. This research concludes that HE's versatility and vast application possibilities are especially significant. Because HE can maintain confidentiality and integrity throughout a wide range of data formats, it is a solid and adaptable encryption technique that can be used in various industries, such as government, healthcare, and finance. In short, this study not only shows how homomorphic encryption can improve data security but also creates new opportunities for cryptography research and development, creating safer and more effective data management techniques.

7.1 Comparative analysis

Table 3: Comparative analysis for Traditional encryption and HE

Traditional Encryption	Homomorphic Encryption(HE)	HE advantages
Security in data processing required decryption, posing potential security threats.	Security in data processing enables computation on encrypted data, Security is maintained throughout the process	Superior security during data processing
Cloud computing data must be decrypted for processing in the cloud, leading to privacy and compliance issues	Cloud computing data utilisation allows secure outsourcing of computation tasks on encrypted data in cloud environments	Enhanced privacy and utility in cloud computing
Many algorithms are vulnerable to quantum computing attacks	Resistance to quantum threats some HE is resistant to quantum computing attack as it offers future-proof security	Future-proof against emerging quantum threats
Not capable of supporting complex application where encrypted data needs to be processed	Complex application support such as secure electronic voting, maintaining data encryption during processing.	Supports complex scenarios maintaining data privacy.
Simpler and well-understood, making it easier to implement and manage	Highly complex due to advance mathematical operations	Allows more secure and advanced operations.
Less flexible as it requires data decryption for computation	Flexible in terms of computation on data	Flexibility in secure data processing.

In Table 3 we have a comparative analysis table of HE with other traditional algorithm and stated the advantages of HE over traditional algorithm.

8 Conclusion and Future Work

8.1 Conclusion

This research aimed to implement HE on strings, which started with implementing HE on the numbers and then moving forward to strings. The implementation was successfully completed on strings. This research aimed to implement HE on strings, which was successfully achieved. This research also addresses the weakness in traditional encryption techniques by implementing HE and using minimal computer power, which can be adaptable in every environment, making it robust. This project showcases a security layer in data that can be a countermeasure against insider threats and data breaches. This research also

addressed the gap in implementing HE in the database, where the problem was executing SQL queries on the database. In this research, the implementation is not only done in strings. It is also in numbers and Special characters, which makes it robust and can be implemented in the database. The comparative analysis further underscores HE's advantages in terms of enhanced security during processing, resistance to quantum threats, and support for complex applications while acknowledging its computational complexity.

8.2 Future Work

Future research should focus on the implementation in HE in databases and objects. Further exploration into reducing the complexity and enhancing the user-friendliness of HE systems would also broaden its adoption. Additionally, developing advanced HE schemes that are resistant to quantum computing attacks would future-proof data security against emerging threats. The implementation of HE in databases and its integration with current cybersecurity infrastructure offer promising avenues for further exploration, potentially revolutionizing how sensitive data is processed and protected in an increasingly digital world.

9 References

Abdalrdha, Z. K., AL-Qinani, I. H., & Abbas, F. N. (2019). Subject Review : Key Generation in Different Cryptography Algorithm. Baghdad: IJSRSET.

ACAR, A., AKSU, H., ULUAGAC, A. S., & CONTI, M. (2018). A Survey on Homomorphic Encryption Schemes: Theory. Miami: ACM Computing Surveys.

Alaya, B., Laouamer, L., & Msilini, N. (2023). Homomorphic encryption systems statement: Trends and challenges. Buraidah: ScienceDirect.

Benaissa, A. (2020). *https://blog.openmined.org/build-an-homomorphic-encryption-scheme-from-scratch-with-python/*. Retrieved 12 11, 2023, from https://blog.openmined.org/build-an-homomorphic-encryption-scheme-from-scratch-with-python/

Elmrabit, N., Yang, S.-H., & Yang, L. (2015). Insider Threats in Information Security. *IEEE.* Glasgow, UK: IEEE.

Gangula, S. (2022). Computation of number using Homomorphic Encryption. Dublin: NCI.

Hellman, M. E. (1978). AN OVERVIEW OF PUBLIC KEY CRYPTOGRAPHY. IEEE .

NETO, N. N., MADNICK, S., PAULA, A. M., & BORGES, N. M. (2021). Developing a Global Data Breach Database and the Challenges Encountered. *ACM Journal of Data and Information Quality.* MIT School of Engineering: ACM .

Ogburn, M., Turner, C., & Dahal, P. (2013). Homomorphic Encryption. *ScienceDirect.* Baltimore: ScienceDirect.

Patel, S. (2023). Evaluating the use of Homomorphic Encryption for secure data processing in cloud networks. Dublin: NCI.

Petrosyan, A. (2023). *Data breaches worldwide - Statistics & Facts*. Retrieved 12 11, 2023, from https://www.statista.com/topics/11610/data-breaches-worldwide/#topicOverview

S, R., B, V., Mehta, A. B., & Honnavalli, P. B. (2022). Homomorphic Encryption Approach for String. *IEEE.* Karnataka: IEEE.

S.Suguna, Dr.V.Dhanakoti, & Manjupriya, R. (2016). A STUDY ON SYMMETRIC AND ASYMMETRIC KEY ENCRYPTION. Kattankulathur: IRJET.

Sectara. (2023). *Addressing insider threats with a cyber security risk matrix*. Retrieved 12 11, 2023, from https://medium.com/@sectara/addressing-insider-threats-with-a-cyber-security-risk-matrix-42da24889a97

TEK˙IN, E. N. (2023). Homomorphic Encryption: A Comprehensive Study of Types, Techniques, and Real-world Applications. Ankara: Middle East Technical University.

Vemula, S., Kovvur, R. M., & Marneni, D. (2023). Algorithms for Implementing Repeated Homomorphic Operations on Restricted Data Type. *SICTIM.* Hyderabad: SICTIM.

Yang, Y. (2012). Evaluation of Somewhat Homomorphic Encryption Schemes. Cambridge: MIT.