International Journal of Scientific Research in Engineering and Management (IJSREM)
Volume: 09 Issue: 04 | April - 2025 SJIF Rating: 8.586 ISSN: 2582-3930

Securing Cloud Storage Using Homomorphic Encryption

B M Sahana
Department of Computer Science and Engineering (Cyber Security)
Sri Shakthi Institute of Engineering and Technology
Coimbatore-India
sahanabm23cys@srishakthi.ac.in

Rohith S
Department of Computer Science and Engineering (Cyber Security)
Sri Shakthi Institute of Engineerin
and Technology
Coimbatore-India
rohiths23cys@srishakthi.ac.in

Sandhiya C
Department of Computer Science and Engineering (Cyber Security)
Sri Shakthi Institute ofEngineering and Technology
Coimbatore-India
sandhiyac23cys@srishakthi.ac.in

Padalingam
Department of Computer Science and Engineering (Cyber Security)
Sri Shakthi Institute of Engineering and Technology
Coimbatore-India
padalingams23cys@srishakthi.ac.in

Mr. P Balaji
Associate Professor
Department of Computer Science and Engineering (Cyber Security)
Sri Shakthi Institute of Engineering and Technology
balajipcys@siet.ac.in

Abstract — This paper focuses on a secure, efficient solution for processing, encrypting, compressing, and storing files in the cloud using Python. The system starts by converting a user-provided text file into ASCII values, which are then encrypted using the CKKS (Cheon-Kim-Kim-Song) homomorphic encryption scheme provided by the TenSEAL library. This encryption ensures that the data remains confidential and protected throughout the entire process. Once encrypted, the file is compressed using Python's gzip module, reducing its size for faster uploads and minimizing storage space. The final step uploads the compressed file to Dropbox, ensuring secure cloud storage. By combining encryption, compression, and cloud integration, this project provides a robust method for securely managing sensitive data, making it ideal for scenarios where both privacy and storage efficiency are paramount. The seamless flow from file conversion to cloud storage offers a practical solution for securely handling large volumes of data.

Keywords — Cloud computing, Security, Cloud storage, Sensitive data, Homomorphic encryption.

1 INTRODUCTION

In today's digital age, the security of sensitive data is of paramount importance, especially with the rise of cloud computing, where vast amounts of personal and business data are stored and processed remotely. The challenge of maintaining data confidentiality, integrity, and privacy while leveraging the benefits of cloud storage requires advanced security techniques. This project addresses these concerns by integrating homomorphic encryption, compression, and cloud storage to provide a comprehensive and secure solution for data processing and storage. The primary objective is to ensure that sensitive data remains encrypted and protected throughout its lifecycle—from the initial processing phase to final storage in the cloud.

At the heart of this project is homomorphic encryption, specifically the CKKS (Cheon-Kim-Kim-Song) scheme, which allows computations to be performed on encrypted data without decrypting it. In traditional encryption methods, data must be decrypted before it can be processed, exposing it to potential risks. However, homomorphic encryption allows operations to be carried out directly on the encrypted data, ensuring that sensitive information remains private even while it is being manipulated or analyzed. This technique is particularly valuable in cloud environments, where data is often processed by third-party servers. By using the CKKS encryption scheme, the project ensures that even if data is accessed during processing, it remains unreadable and secure.

Once the data is encrypted, the program moves to the next critical phase: compression. Encrypted data tends to be much larger than its unencrypted counterpart, which can create storage and transmission challenges. To address this, the encrypted data is compressed using Python’s gzip module. Compression reduces the size of the encrypted file, optimizing storage efficiency and minimizing upload times. This step is essential for cloud storage, as it allows for faster data transfer and helps conserve storage space, particularly when dealing with large files. By compressing the data before uploading it to cloud storage, the project ensures that the encrypted files are both space-efficient and more manageable in terms of upload time.

Finally, the compressed, encrypted file is uploaded to a cloud storage platform, such as Dropbox, using the Dropbox API. Cloud storage has become an essential tool for modern data management, offering scalable, remote storage that can be accessed from anywhere. However, this also raises security concerns, as cloud data is susceptible to unauthorized access and breaches. To mitigate these risks, the project ensures that the data remains encrypted throughout the entire process, even during transfer to and storage in the cloud. By integrating encryption, compression, and cloud storage, this project provides a secure and efficient solution for managing sensitive data, ensuring that it remains protected from potential threats while benefiting from the convenience and scalability of cloud platforms.

2 LITERATURE REVIEW

Homomorphic encryption (HE) has become a promising solution for securing data during computation. It allows computation on encrypted data without revealing sensitive information, thus ensuring privacy. Fully Homomorphic Encryption (FHE) was a significant breakthrough introduced by Gentry (2009), which enabled arbitrary computation on encrypted data, though its inefficiency in practice remained a key limitation. Since then, advancements in leveled homomorphic encryption (LHE) and schemes such as CKKS (Cheon-Kim-Kim-Song) have improved its feasibility for applications like privacy-preserving machine learning and data analytics. Recent studies such as Bachem et al. (2021) discuss the use of CKKS for approximate computations over encrypted data, highlighting its use in real-world applications where approximate results are acceptable. Kaya et al. (2023) further advance this by demonstrating how the TenSEAL library has simplified the integration of homomorphic encryption for machine learning workflows, making it easier for data scientists and engineers to apply encryption in secure data processing. Their work shows that CKKS can be used effectively for encrypted data without compromising computational accuracy. Additionally, Mohammad et al. (2022) introduced optimizations for homomorphic encryption schemes by leveraging GPU acceleration and parallel computing to improve performance and reduce overhead, a significant concern in the practical application of HE. These innovations are critical for implementing privacy-preserving computations at scale.

As cloud computing and storage services continue to grow, concerns about data privacy and security have emerged. In cloud environments, data must be protected against unauthorized access, breaches, and data leakage. Zhang et al. (2022) investigate the current state of cloud security and suggest that end-to-end encryption and access control mechanisms are essential for ensuring confidentiality and privacy. They emphasize the importance of client-controlled encryption and propose models for securing data without relying entirely on cloud service providers. Xie et al. (2021) introduced a hybrid encryption model combining symmetric and asymmetric encryption to balance performance and security in cloud environments. This approach ensures that cloud users' sensitive data remains confidential during storage and processing. One of the significant contributions to cloud security is the integration of homomorphic encryption, which provides privacy-preserving computations. Li et al. (2023) discuss how HE can be applied in cloud computing environments to prevent cloud service providers from accessing raw data while enabling computations. By utilizing HE, data remains encrypted throughout its lifecycle, from uploading to processing. However, key management in cloud environments remains an area of concern. Zhao et al. (2024) address this issue by exploring secure key management protocols that enhance the security of encryption keys, preventing unauthorized access to the keys that are critical for decrypting data. Their research highlights that even if cloud service providers are compromised, the data remains

secure if the keys are properly managed. Data compression is another essential element in cloud storage systems, as it helps reduce the volume of data that needs to be stored and transmitted, thus improving efficiency. Compressed files consume less space, resulting in lower storage costs and faster upload/download speeds. However, when combined with encryption, data compression becomes more challenging due to the randomness introduced by encryption algorithms. Encrypted data typically has high entropy, which makes it resistant to compression, as there is little redundancy in the encrypted data to exploit.

Wang et. al.,(2021) discuss various methods of compressing encrypted data and the difficulties that arise when trying to apply traditional compression techniques, such as gzip, to encrypted files. They suggest that encryption often reduces the effectiveness of standard compression algorithms due to the lack of patterns in the encrypted data. To address this, they propose using advanced techniques such as delta compression and dictionary-based compression specifically designed for encrypted data. These methods aim to find patterns or redundancies within the encrypted data and compress it without compromising its security. Khan et al. (2022) further expand on these methods by suggesting that a more tailored approach to compression is needed for encrypted files. Their research focuses on optimizing compression algorithms based on the encryption scheme used, as different types of encryption introduce different challenges for compression. They show that applying such specialized algorithms can help reduce the size of encrypted files while maintaining data security, making cloud storage more efficient and cost-effective. Furthermore, Singh et al. (2023) propose an integrated approach that combines both encryption and compression to enhance the performance of cloud storage systems. Their work demonstrates that by compressing data before encrypting it, cloud systems can achieve more efficient storage and faster data transmission. Additionally, they discuss how incorporating these compression techniques with end-to-end encryption ensures that the data remains secure during both storage and transfer. This integration of encryption and compression methods helps balance the trade-offs between security and efficiency, offering practical solutions for cloud storage providers looking to enhance their services.

The combination of homomorphic encryption, cloud storage security, and data compression has proven to be an effective approach for addressing the challenges of securely processing and storing sensitive data in cloud environments. CKKS-based encryption offers a promising solution to enable privacy-preserving computations on encrypted data, while the integration of compression techniques helps reduce storage and transmission costs without compromising security. Recent advancements in cloud key management (Zhao et al., 2024) and optimizations for homomorphic encryption (Mohammad et al., 2022) contribute to making these solutions more practical and scalable. As the demand for secure and efficient cloud storage continues to grow, the integration of these technologies will play a crucial role in ensuring the confidentiality, integrity, and privacy of data in the cloud.

3 PROPOSED SYSTEM

This research investigates the vulnerabilities associated with storing and processing encrypted data in cloud environments, focusing on compliance with data protection laws (e.g., GDPR, HIPAA, PCI-DSS). By leveraging Python and the TenSEAL library, researchers evaluate how homomorphic encryption methods can align with regulatory requirements, highlighting compliance measures for secure cloud storage. The study also examines legal precedents and enforcement actions related to data breaches involving unencrypted cloud data, understanding the potential penalties for organizations that fail to secure sensitive data.

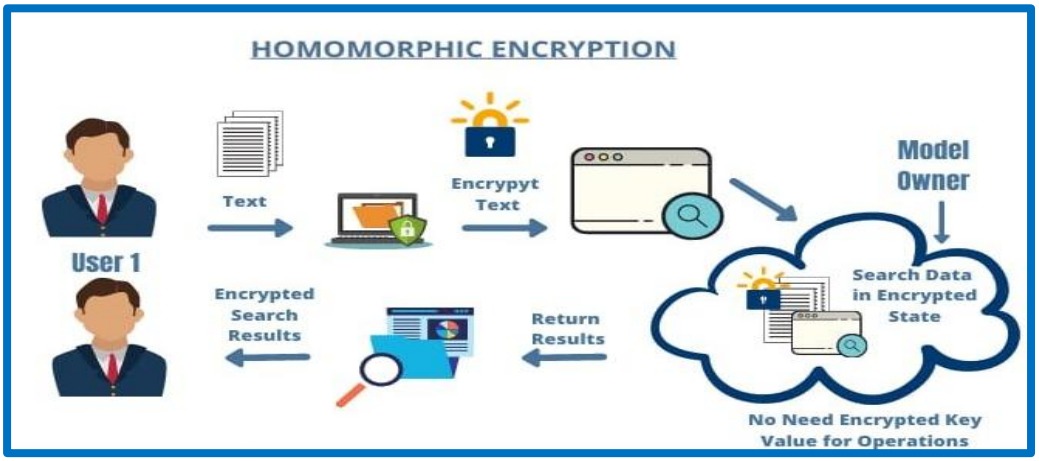

Fig 3.1 Working of homomorphic encryption

The Fig 3.1 illustrates the process of Homomorphic Encryption, a method for securely processing data while keeping it encrypted. In this process, a user (User 1) begins with plain text data that they wish to keep confidential. This data is

encrypted on the user's side, converting it into ciphertext, which protects it from unauthorized access. The encrypted data is then sent to a third party, referred to as the Model Owner (often a cloud service provider or computational model owner), for processing. The unique aspect of homomorphic encryption is that the Model Owner can perform operations, such as searches or computations, directly on the encrypted data without needing to decrypt it, ensuring that the content remains confidential throughout. The Model Owner does not require the encryption key, which means they can process the data without compromising its privacy. After processing, the encrypted results are sent back to the user, who then decrypts the results to access the meaningful output. This approach provides significant privacy and security benefits by allowing computations on sensitive data without exposing it, and by ensuring that the Model Owner never accesses the raw data or the decryption key.

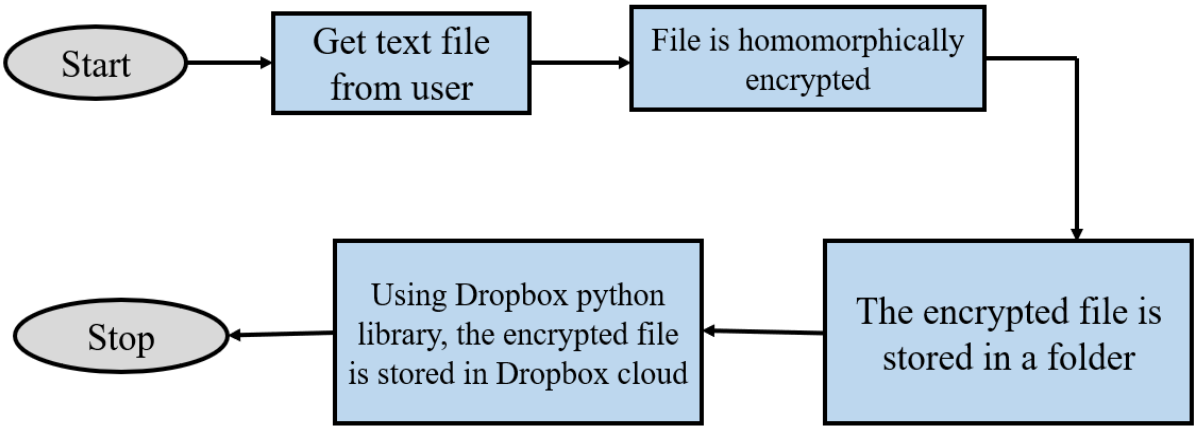

Fig 3.2 Securing cloud storage using homomorphic encryption

The Fig 3.2 illustrates the process of securing cloud storage using Homomorphic Encryption. It begins with a user providing a text file containing data that they want to securely store in the cloud. Once the file is received, it undergoes homomorphic encryption, transforming the file's content into an encrypted format. Homomorphic encryption is a cryptographic technique that allows data to remain encrypted even as it is stored, transmitted, or processed, ensuring privacy and security. After the file is encrypted, it is saved in a designated folder. This step organizes the encrypted data and prepares it for cloud storage. Next, the encrypted file is uploaded to a cloud storage platform—specifically Dropbox—using the Dropbox Python library. This integration with Dropbox allows for automated uploading and storage of the encrypted file in a cloud environment, enhancing the security of data storage by keeping the information encrypted at all stages. With homomorphic encryption, even if unauthorized parties access the stored file, they would be unable to interpret the data without the appropriate decryption keys. This secure storage process ensures data confidentiality, as the encryption remains intact throughout. The flowchart concludes after the file is successfully stored in Dropbox, showing a simple yet effective approach to secure sensitive information using cloud storage with homomorphic encryption.

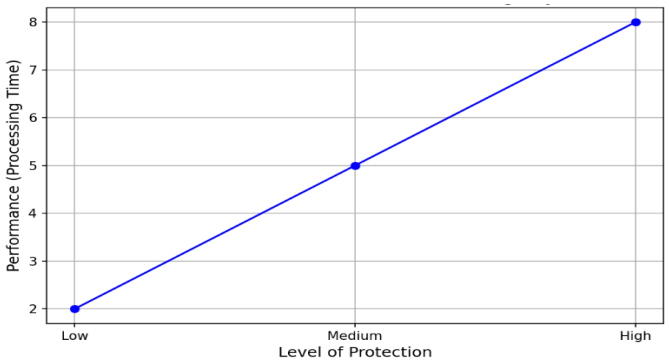

Fig 3.3 Performance over level of protection graph

The Fig 3.3 illustrates the relationship between escalating protection levels and corresponding processing times in a cloud storage environment, as demonstrated by project outcome. The process begins with converting data to ASCII values, offering minimal security with minimal processing overhead, represented by the lowest point on the graph. The next phase, applying CKKS homomorphic encryption, significantly enhances security but incurs additional computational demands, leading to a moderate increase in processing time. Finally, compressing the encrypted data and uploading it to Dropbox achieves the highest level of security at the cost of the longest processing time. This trend reflects a fundamental trade-off in secure cloud storage systems: as data protection measures become more robust, the associated computational and processing requirements increase. The graph underscores the balance that must be maintained between data security and system performance in cloud environments, particularly when utilizing advanced encryption techniques.

4 EXPERIMENTAL RESULT

The first step of the workflow is to take an input text file provided by the user (Fig 4.1) and convert its content into a series of ASCII values (Fig 4.2). This is accomplished by reading the file line by line and then converting each character in the line to its corresponding ASCII value using Python's built-in ord() function. When the user is prompted to enter the path of the text file, the program reads the file's contents into memory. It processes each line of the file, converts every character into its ASCII equivalent, and stores the resulting ASCII codes as strings. These strings are then written to an output file, output_ascii.txt, with each line of ASCII values written on a new line. This file serves as a plaintext intermediary that holds the ASCII representation of the original content. This step is particularly useful for applications that require working with numerical data rather than raw text, such as cryptographic processes or data compression. The conversion to ASCII essentially maps each character of the original text to its numerical representation, and by storing this mapping, the program lays the groundwork for the encryption and compression processes that follow.

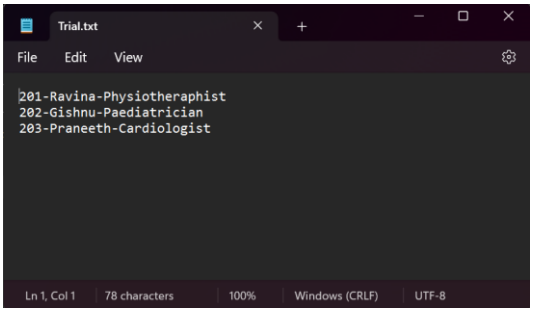

Fig 4.1 Input Text File (.txt format)

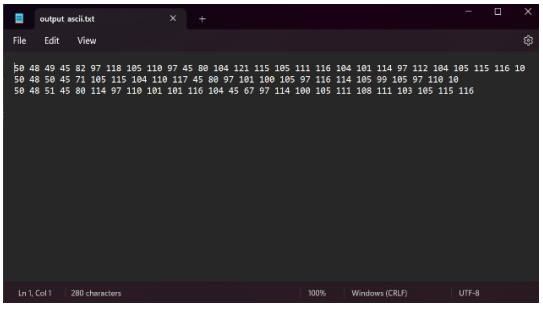

Fig 4.2 ASCII Conversion of Input text file

The encrypt_file function follows, employing the CKKS homomorphic encryption scheme, as implemented in the TenSEAL library, to securely encrypt the ASCII values. Homomorphic encryption supports computation on encrypted data, making it invaluable for privacy-preserving data analysis, particularly in untrusted environments such as cloud computing. In this function, each ASCII integer is converted to a floating-point number, as required by CKKS. A TenSEAL encryption context is established, defining key parameters such as the scale, which regulates precision in encrypted computations. Each ASCII value is then encrypted individually, serialized, and encoded in Base64 to facilitate text storage. The resulting encrypted and encoded values are written line by line to a new file (Fig 4.3), preserving the structure while transforming the data into a securely encrypted form. This encrypted file is now suitable for secure storage and transmission, with its contents obscured from unauthorized access.

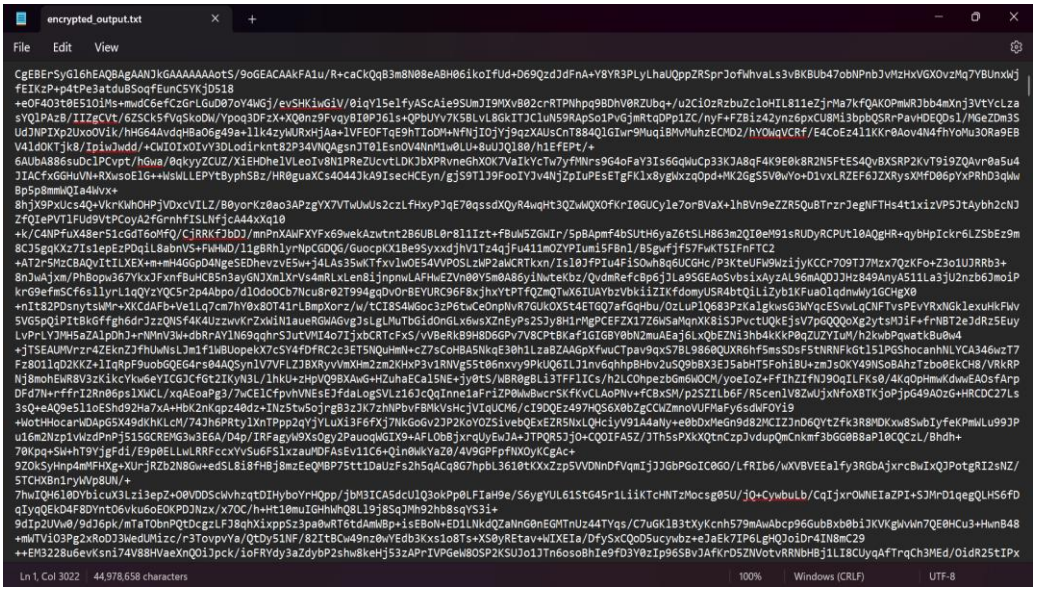

Fig 4.3 Homomorphic encryption of converted text file

Next, the compress_encrypted_file function optimizes the encrypted file for storage and transfer by removing extraneous whitespace and compressing it into a .gz format. This compression step is not merely a matter of storage efficiency but is also critical for minimizing file size and transmission time when uploading to the cloud. The function reads each line of the encrypted file, strips unnecessary whitespace, and removes blank lines to produce a compact version. The resulting compressed file significantly reduces storage requirements and facilitates faster network transmission (Fig 4.4), which is particularly advantageous when handling large datasets or constrained network environments.

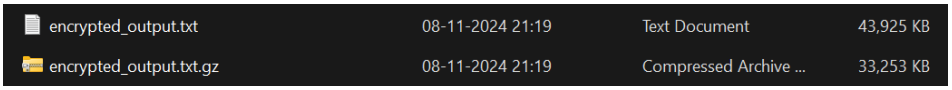

Fig 4.4 Encrypted file compression

The upload_to_dropbox function completes the workflow by securely transferring the compressed, encrypted file to Dropbox. Utilizing the Dropbox API, the function authenticates with an access token, allowing access to the user's Dropbox account while ensuring secure file handling. Before uploading, the function verifies the file's existence, thereby preventing errors due to missing or inaccessible files. The file is then uploaded to Dropbox's root directory, retaining its local filename. This cloud-based storage of the encrypted file provides a secure and accessible solution for long-term data retention, enabling users to manage sensitive files securely from any location.

To conclude, this Python program provides an effective and secure workflow for handling sensitive data by integrating ASCII conversion, homomorphic encryption, file compression, and cloud storage. By encoding data to ASCII and then applying CKKS homomorphic encryption with the TenSEAL library, it ensures that data confidentiality is maintained even when stored in cloud environments. Compressing the encrypted data with gzip enhances storage efficiency and reduces upload times, while the Dropbox integration allows for convenient and secure cloud storage. Each function operates together cohesively, ensuring sensitive information remains protected throughout the entire process. This program serves as a powerful tool for anyone needing to safeguard data, with potential for further enhancements such as advanced security measures and more comprehensive error handling. As a result, it provides a versatile and reliable solution for secure data management from local environments to cloud storage.

5 CONCLUSION :

In conclusion, this paper introduces a comprehensive, secure data-handling system that combines ASCII conversion, homomorphic encryption, file compression, and cloud storage using a Python-based workflow. Designed with data privacy at its core, this system ensures security across all stages, beginning with ASCII transformation and progressing through CKKS encryption with the TenSEAL library. By enabling encrypted computations, CKKS encryption preserves data confidentiality, making it suitable for privacy-preserving applications. The integration of gzip compression reduces file size, enhancing storage efficiency, while Dropbox integration provides a secure and accessible cloud storage solution.

This paper has broad implications for fields requiring secure data handling, including remote data analytics, cloud-based processing, and privacy-preserving machine learning. Future research directions could focus on optimizing the uploading speed, particularly through alternative compression algorithms that maintain encryption integrity while reducing latency. Additionally, integrating parallel processing for file handling, or exploring more direct upload protocols compatible with encrypted data, could significantly improve performance. Further improvements could also include optimizing encryption schemes for computational efficiency and expanding compatibility with other cloud platforms. This project lays a foundation for secure, scalable data management solutions, addressing key challenges in cybersecurity and secure cloud storage.

REFERENCES

[1] Potey, Manish & Dhote, Chandrashekhar & Sharma, Deepak. (2016). Homomorphic Encryption for Security of Cloud Data. Procedia Computer Science. 79. 175-181. 10.1016/j.procs.2016.03.023.

[2] Gentry, Craig. (2009). Fully Homomorphic Encryption Using Ideal Lattices. Proceedings of the Annual ACM Symposium on Theory of Computing. 9. 169-178. 10.1145/1536414.1536440.

[3] Bhargavi, Peyakunta & Srilakshmi, K.. (2020). Efficient dual compression and encryption technique for enhanced cloud security. Journal of Green Engineering. 10. 8721-8735.

[4] Wiryen, Yancho & Vigny, Noumsi & Mvogo Ngono, Joseph & Fono, Louis. (2024). Leveraging TenSEAL:. International Journal of Information Security and Privacy. 18. 1-17. 10.4018/IJISP.356402.

[5]Wainakh, Asndar. (2018). Homomorphic Encryption for Data Security in Cloud Computing.

[6] Gladston, Angelin & Naveenkumar, S. & Sanjeev, K. & Gowthamraj, A.. (2024). An Accelerator to Additive Homomorphism to Handle Encrypted Data. International Journal of Business Data Communications and Networking. 19. 1-25. 10.4018/IJBDCN.341589.

[7] Kumar, Shantanu. (2024). Enhancing cloud storage efficiency and accessibility with artificial intelligence: A comprehensive review. International journal of advanced research in engineering & technology.15. 183-196. 10.17605/OSF.IO/9DC2J.

[8] Qasim, Nameer & Bodnar, Natalia & Salman, Hayder & Mustafa, Salama & Rahim, Fakher. (2024). Data Management Challenges and Solutions in Cloud-Based Environments. Radioelectronics. Nanosystems. Information Technologies.16. 157-170. 10.17725/j.rensit.2023.16.157.

[9] Bhat, Aaqib & Kumar, Rajiv. (2024). Efficient Hybrid Encryption Algorithm for Securing Data in Cloud Environment. 10.21203/rs.3.rs-4233929/v1.

[10] Sengupta, Shubhashis & Kaulgud, Vikrant & Sharma, Vibhu Saujanya. (2011). Cloud Computing Security--Trends and Research Directions. Proceedings - 2011 IEEE World Congress on Services, SERVICES 2011. 524-531. 10.1109/SERVICES.2011.20

[11] Xiao, Zhifeng & Xiao, Yang. (2013). Security and Privacy in Cloud Computing. Communications Surveys & Tutorials, IEEE. 15. 843-859. 10.1109/SURV.2012.060912.00182.

[12] Vankudoth, Biksham & Vasumathi, D.. (2017). Homomorphic Encryption Techniques for securing Data in Cloud Computing: A Survey. International Journal of Computer Applications. 160. 1-5. 10.5120/ijca2017913063.

[13] Mohialden, Yasmin & Mahmood Hussien, Nadia & Salman, Saba & Aljanabi, Mohammad. (2023). Secure Federated Learning with a Homomorphic Encryption Model. 4. 001-007. 10.47667/ijpasr.v4i3.235.

[14] Ananthakrishna, V. & Yadav, Dr. (2023). Advancements in Cloud Security: An Enhanced Auth Privacy Chain-Based Hybrid Encryption Technique for Scalability. Migration Letters. 20. 485-497. 10.59670/ml.v20iS13.6478.

[15] Huang, Chun-Ting & Huang, Lei & Qin, Zhongyuan & Yuan, Hang & Zhou, Lan & Varadharajan, Vijay & Kuo, C.-C. Jay. (2014). Survey on securing data storage in the cloud. APSIPA Transactions on Signal and Information Processing. 3. 10.1017/ATSIP.2014.6.

[16] Lee, Joon-Woo & Kang, Hyungchul & Lee, Yongwoo & Choi, Woosuk & Eom, Jieun & Deryabin, Maxim & Lee, Eunsang & Lee, Junghyun & Yoo, Donghoon & Kim, Young-Sik & No, Jong-Seon. (2022). Privacy-Preserving Machine Learning With Fully Homomorphic Encryption for Deep Neural Network. IEEE Access. 10. 30039-30054. 10.1109/ACCESS.2022.3159694.