Advancing Data Security in Cloud Computing: A Comprehensive Exploration of Quantum-Secure Variant of Fully Homomorphic Encryption Technique

Dr. I. Carol,
Assistant Professor,
Department of Information Technology,
St.Joseph's College,
Tiruchirappalli-620 002

Abstract

The increasing reliance on Cloud Computing for large-scale data processing has raised significant concerns about data security and privacy, especially in the face of emerging quantum threats. Traditional Fully Homomorphic Encryption (FHE) schemes, while enabling secure computations on encrypted data, face vulnerabilities against quantum attacks due to their reliance on classical cryptographic hardness assumptions. This research explores a Quantum-Secure Variant of Fully Homomorphic Encryption (QFHE) that leverages Module-Learning With Errors (Module-LWE) and Ring-LWE to provide post-quantum security while maintaining efficient homomorphic computations. The proposed QFHE scheme integrates lattice-based cryptography with verifiability mechanisms, ensuring both computational integrity and resistance to quantum adversaries. Additionally, optimizations in key switching and bootstrapping techniques enhance efficiency, making the model viable for real-world cloud applications. By securing encrypted data operations against both classical and quantum threats, QFHE presents a forward-thinking solution for enterprises seeking robust, future-proof data security in Cloud Computing environments.

Keywords: Cloud Computing, Data Security, Quantum-Secure Fully Homomorphic Encryption.

1. Introduction

As cloud computing continues to evolve, ensuring strong data security remains a critical challenge. Traditional security methods often fall short against emerging threats such as data breaches and unauthorized access. Advancements in encryption, access control, and AI-driven anomaly detection have enhanced cloud security, improving data confidentiality and integrity. Implementing multi-layered security frameworks and compliance-driven strategies strengthens protection against cyber threats. This paper examines innovative approaches to securing cloud environments, addressing key vulnerabilities, and optimizing data protection measures [1].

This research explores a QFHE a cryptographic framework designed to perform secure computations on encrypted data while remaining resistant to quantum adversaries. By leveraging lattice-based cryptography, specifically Module-LWE and Ring-LWE, QFHE ensures robust security against quantum attacks while enabling efficient homomorphic operations. Unlike conventional Fully Homomorphic Encryption (FHE), which relies on classical hardness

assumptions, QFHE incorporates post-quantum cryptographic techniques to future-proof cloud data security.

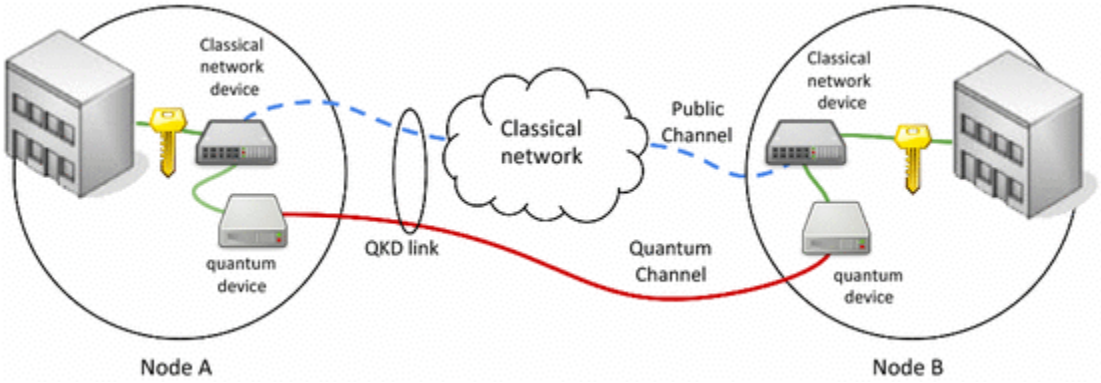

Fig.1. General Data encryption system architecture

Figure 1 illustrates a cloud security framework where users interact with encrypted data using quantum-resistant encryption and decryption. The proposed QFHE algorithm leverages lattice-based cryptography, ensuring efficiency and resilience against quantum attacks. By integrating post-quantum techniques like Module-LWE and Ring-LWE, QFHE enables secure computations on encrypted data without exposure to threats. This approach strengthens cloud security, allowing enterprises to adopt cloud computing while mitigating risks from both classical and quantum adversaries.

2. Literature Review

Ameur, Y., Bouzefrane, S., & Thinh, L. V. (2023) further discuss the security aspect, stating that current encryption algorithms are efficient but resource-intensive, making them costly and time-consuming to manage. Moreover, traditional encryption methods make it impossible to process data without first decrypting it. Specifically, conventional public-key encryption requires data to be decrypted before it can be analyzed or manipulated. In contrast, homomorphic encryption allows data to remain encrypted while being processed, enabling users or third parties, such as cloud providers, to perform operations on encrypted data without revealing its contents. the high performance and robust data processing capabilities of cloud computing, externalizing data to cloud platforms has become an inevitable trend in the digital landscape today. However, ensuring the security and privacy of data remains a significant challenge. To address this concern, a multi-cloud platform is proposed to enhance both privacy and high availability of data. This multi-cloud platform integrates public, private, and managed clouds through a unified user interface. Data hosted on the cloud is distributed across various data centers within the multi-cloud environment, based on cloud reliability and the sensitivity of the data [2].

P. Ora and P. R. Pal addressees this issue by combining RSA Partial Homomorphic encryption with MD5 hashing. RSA Partial Homomorphic encryption allows cloud servers to perform computations on encrypted data without decrypting it, ensuring confidentiality. MD5 hashing, on the other hand, is used to verify data integrity by generating a unique hash of the encrypted data. The approach involves encryption, data uploading, hashing, and verification, ensuring that data remains secure and unaltered [3]. Tang et al. (2025) proposed a Threshold Quantum

Homomorphic Encryption (TQHE) scheme, based on the Shamir secret sharing protocol, which enables multiple evaluators (ranging from 3 to 5) to collaboratively perform computations on encrypted quantum data. This scheme allows each evaluator to carry out arbitrary single-qubit gate operations on the encrypted data while maintaining the overall security of the system. A key contribution of this work is the flexibility of the scheme, as it enables multiple evaluators with independent quantum computing resources to jointly evaluate encrypted quantum data. This opens up the possibility for more complex and scalable computations on quantum networks compared to single-evaluator systems. The authors provide a specific example using a (3, 5)-threshold configuration, demonstrating the feasibility and correctness of the approach through simulations on the IBM quantum computing cloud platform.The security of the proposed scheme is rigorously analyzed, covering aspects such as encryption/decryption private keys, quantum state sequences during transmission, and the final computation results. This thorough analysis ensures that the proposed TQHE scheme maintains the confidentiality and integrity of quantum data while enabling flexible collaborative computations [4].

Hamza et al., (2022), the authors provide an overview of FHE algorithms applied to Big Data. They present a security framework for Big Data analysis, integrating HE to ensure privacy during computation. The paper compares various homomorphic encryption tools, evaluating their performance in terms of scalability, efficiency, and resource usage. The authors highlight trade-offs between security and computational cost when choosing HE techniques for Big Data applications. The study also identifies key research challenges and future opportunities for optimizing HE algorithms, particularly in integrating privacy-preserving techniques with machine learning for enhanced Big Data processing. This work sheds light on the potential of HE for secure Big Data analytics, offering practical insights into its use and challenges [5].

Kim and Yun (2021) proposed a new security notion for homomorphic authenticated encryption, which unifies data privacy and authenticity in a simpler and stronger way than previous definitions. The paper presents the first construction of fully homomorphic authenticated encryption, combining fully homomorphic encryption with two homomorphic authenticators one fully homomorphic and one OR-homomorphic. This construction ensures the security of data privacy and authenticity, requiring the encryption to be indistinguishable under chosen plaintext attacks and the authenticators to be unforgeable under selectively chosen plaintext queries. Additionally, the authors propose a multi-dataset fully homomorphic authenticator scheme, which enhances efficiency by supporting amortized performance and satisfying security requirements. This work advances the field by providing a robust construction for fully homomorphic authenticated encryption, addressing both security and efficiency concerns in homomorphic computations on multiple datasets [6].

Brakerski (2018) introduced a Quantum Fully Homomorphic Encryption (QFHE) scheme that allows quantum-efficient computations on both classical and quantum encrypted data, similar to classical FHE. The security of the scheme relies on the Learning With Errors (LWE) problem with polynomial modulus, which aligns with the best-known security assumptions for classical FHE. To support unbounded computation depth, the scheme requires a circular security assumption, which is also used in multi-key classical FHE.Brakerski also highlights the connection between evaluating quantum gates and the circuit privacy property in classical FHE, offering a pathway to constructing QFHE using classical FHE techniques. This work brings

quantum encryption closer to practical implementation by leveraging existing classical FHE schemes for quantum data [7].

Mittal and Ramachandran (2021) presented a systematic review of Fully Homomorphic Encryption (FHE) research over the past decade. As cloud computing grows and big data becomes increasingly prevalent, confidentiality and security challenges, especially in public cloud environments, have sparked interest in advanced encryption models. HE allows computations on encrypted data without decryption, offering a potential solution to privacy concerns.The review focuses on recent developments in FHE, discussing various algorithms such as Lattice-based, integer-based, Learning With Errors (LWE), Ring Learning With Errors (RLWE), and Nth degree Truncated Polynomial Ring Units (NTRU). These methods are examined for their role in enhancing the security and efficiency of cloud-based applications. Additionally, the paper highlights the challenges and gaps in FHE research, particularly in terms of performance and scalability, and provides insights into future research directions for more effective FHE models in the cloud sector.The work offers valuable contributions to the field, focusing on how FHE can strengthen data security and privacy in cloud computing, while also identifying areas requiring further exploration [8].

Mustafa et al. (2020) addressed the vulnerabilities of the conventional RSA algorithm in the context of IoT-based cloud applications. Due to advancements in quantum computing, traditional RSA can be easily compromised, highlighting the need for post-quantum cryptographic methods. To tackle this, the paper proposes a lattice-based RSA (LB-RSA) algorithm, which incorporates quantum-resistant features to secure shared data in IoT environments.The proposed LB-RSA technique is validated with a 60-dimensional key size of approximately 1.152 × 10^5 bits, achieving a generation time of 0.8 hours. The security of the algorithm is confirmed through testing with AVISPA, ensuring robustness against potential intruders. When compared to existing cryptographic methods, LB-RSA demonstrates superior security for data sharing. The empirical results suggest that the lattice-based approach not only ensures post-quantum security but also outperforms other contemporary techniques in securing communication in IoT-based cloud systems [9].

Sanon et al. (2024) explored the potential of Fully Homomorphic Encryption (FHE) to address the limitations of traditional encryption methods in mobile communication. As wireless communication systems evolve to meet increasing data processing demands, FHE presents a promising solution by enabling computations on encrypted data without decryption, ensuring both security and privacy. The paper identifies key applications of FHE and critically evaluates its integration into mobile communication systems. Practical demonstrations, such as secure network slicing, highlight FHE's potential to enhance network security and privacy. Despite its promise, the authors emphasize the need for further research to fully harness FHE's capabilities in mobile communication and develop secure, privacy-aware networks for the future [10].

3. Materials and methodology

Data encryption in the cloud is a fundamental practice for ensuring the security and confidentiality of sensitive information. With the increasing use of cloud services, where data is

stored and processed remotely, encryption serves as a crucial defense mechanism against unauthorized access, data breaches, and cyber-attacks.

Table.1. Table summarizing the key terms in cloud data encryption

Term	Description
Encryption	The process of converting plaintext data into ciphertext to secure it during storage or transmission.
Decryption	The reverse process of encryption, converting ciphertext back into its original plaintext form.
Cipher	An algorithm or set of rules used for encryption and decryption of data (examples include AES, RSA).
Key	A piece of information used within an algorithm to transform plaintext into ciphertext or vice versa.
Symmetric Encryption	A type of encryption where the same key is used for both encryption and decryption.
Asymmetric Encryption	A type of encryption that uses a pair of keys: a public key for encryption and a private key for decryption.
Public Key	A key that is openly shared in asymmetric encryption systems, used for encrypting data.
Private Key	A confidential key in asymmetric systems used for decrypting data, which must be kept secure.
Key Management	The process of managing the creation, exchange, storage, and destruction of cryptographic keys.
End-to-End Encryption	Ensures that data can only be read by the communicating parties, preventing unauthorized access during transit.
Hash Function	A one-way cryptographic function that generates a fixed-size hash value for data integrity verification.
Digital Signature	A cryptographic technique used to verify the authenticity and integrity of a message or document.

This table provides a concise overview of important encryption-related concepts used in securing cloud data [11].

i. Message Digest 5 (MD5) Algorithm:

The MD5 (Message Digest Algorithm 5) is a widely used cryptographic hash function that produces a 128-bit hash value from an input message. It ensures the integrity of data transmitted over potentially insecure channels by generating a unique digest (hash) for the message. If the message is altered, the resulting hash will also change, making it easier to detect tampering. The process involved in generating the MD5 hash is as follows:

- **Initialize Variables:** Four 32-bit variables, typically labeled as A, B, C, and D, are initialized with predefined constants. These variables serve as the initial state for the hash calculation.

- **Padding:** The message is padded to ensure its length is congruent to 448 modulo 512. This is achieved by appending a single '1' bit followed by enough '0' bits, along with the length of the original message (in bits) as a 64-bit representation.

- **Process Message in Blocks:** The padded message is divided into 512-bit blocks. If the message is not already a multiple of 512 bits, the padding step ensures the correct length.

- **Initialize Hash Values:** For each block, initial hash values are set according to the MD5 specification, ensuring a consistent starting point for each round of processing.

- **Process Blocks:** Each 512-bit block is processed through a series of four rounds involving bitwise operations and mathematical transformations. In each round, the data undergoes substitution, permutation, and mixing operations to create diffusion and confusion.

- **Update Hash Values:** After processing each block, the intermediate hash values (A, B, C, and D) are updated. These updates are accumulated through each iteration, refining the hash as the process progresses.

- **Output:** After all blocks have been processed, the final hash values A, B, C, and D are concatenated to form the 128-bit MD5 digest. This digest is the unique hash representation of the original message.

The MD5 algorithm, despite being fast and efficient, is no longer considered secure against collision attacks, where two different inputs produce the same hash. Consequently, it is recommended to use stronger hash algorithms, such as SHA-256, for applications requiring high levels of security [12].

ii. Secure Hash Algorithm (SHA) Algorithm:

The Secure Hash Algorithm (SHA) family of cryptographic hash functions follows a procedure similar to MD5, involving key steps such as padding, block processing, hash initialization, and updating. However, SHA algorithms, particularly the SHA-2 variants, provide significantly stronger security guarantees and are widely used in modern cryptographic applications.

- **Initialization:** The SHA algorithm begins by initializing specific variables based on the chosen variant (e.g., SHA-1, SHA-256). These initial values are predetermined and serve as the starting point for the hashing process.

- **Padding:** To ensure the message is aligned with the algorithm's block size (typically 512 or 1024 bits), the message undergoes padding. This step adds a single '1' bit followed by sufficient '0' bits. Additionally, the original length of the message, in bits, is appended at the end to make the total length a multiple of the block size.

- **Message Division:** Once padded, the message is split into blocks, where each block will be processed separately. The division of the message into blocks ensures the algorithm can handle messages of arbitrary length efficiently.

- **Hash Value Initialization:** Initial hash values are set depending on the specific SHA variant used. These initial values are crucial, as they provide the starting point for the iterative process that generates the final hash.

- **Block Processing:** Each block is processed through multiple rounds, where a series of bitwise operations, modular additions, and logical functions are applied. These operations work together to transform the block, ensuring that even a small change in the input message leads to a dramatically different hash value.

- **Hash Value Update:** After processing each block, the hash values are updated, building upon the work done in previous rounds. This iterative process continues, incorporating the results of each block into the overall hash computation.

- **Final Output:** Once all message blocks have been processed, the final hash values are concatenated to produce the digest. The output length varies by the SHA variant, with SHA-1 producing a 160-bit hash and SHA-256 producing a 256-bit hash.

Security Considerations: SHA-2 variants, such as SHA-256, are preferred over earlier versions like SHA-1 due to their improved resistance to vulnerabilities like collision and preimage attacks. As computational power increases, the SHA-2 family provides a more secure alternative for hashing sensitive data, ensuring robust protection in modern cryptographic systems [13].

iii. AES (Advanced Encryption Standard) Algorithm:

AES (Advanced Encryption Standard) is a widely used symmetric encryption algorithm, built on the Rijndael block cipher. It operates on fixed-size blocks of 128 bits and offers flexible security levels with three key size options: 128-bit, 192-bit, and 256-bit. The number of encryption rounds varies with the key size, providing different layers of security for each configuration. Specifically, AES performs 10 rounds for 128-bit keys, 12 rounds for 192-bit keys, and 14 rounds for 256-bit keys [14].

Encryption Process:

- **Byte Substitution (SubBytes)**: The 128-bit input block is divided into bytes and substituted using a substitution box (S-Box), which replaces each byte with a corresponding value from a predefined table. This transformation introduces non-linearity to the data.

- **Shift Row Transformation (ShiftRows)**: The rows of the state matrix are shifted left in a circular manner. The first row remains unchanged, while the second, third, and fourth rows are shifted by 1, 2, and 3 positions, respectively. This step helps to mix the data and create diffusion across the state matrix.

- **Mix Column Transformation (MixColumns)**: This step mixes the columns of the state matrix to provide further diffusion. Each column is transformed using a mathematical operation that combines the bytes in the column, making it harder to reverse-engineer the original input.

- **Add Round Key (AddRoundKey)**: The state matrix is XORed with the round key derived from the original encryption key through a key expansion process. This step introduces the encryption key into the transformation, adding another layer of security.

Decryption Process:

- **Inverse Byte Substitution (InvSubBytes)**: The inverse of the byte substitution step is applied using an inverse S-Box, reversing the substitutions made during encryption.

- **Inverse Shift Row Transformation (InvShiftRows)**: The rows of the state matrix are shifted in the opposite direction (right circular shift) to undo the row shifts applied during encryption.

- **Inverse Mix Column Transformation (InvMixColumns)**: The inverse of the mix column operation is applied to reverse the mixing of data. This step ensures the state matrix returns to its original form before encryption.

- **Add Round Key (Inverse)**: Finally, the round keys are applied in reverse order, using the key expansion process in reverse to recover the original encryption key.

AES is designed to ensure secure encryption and decryption of data. Its use of multiple rounds of transformation (substitution, shifting, mixing, and key addition) provides robust security. The number of rounds is determined by the key size: 10 rounds for a 128-bit key, 12 rounds for a 192-bit key, and 14 rounds for a 256-bit key. These processes collectively make AES one of the most widely trusted encryption algorithms in modern cryptography.

iv. RSA (Rivest-Shamir-Adleman) Algorithm

The RSA algorithm is a widely recognized public-key cryptosystem used for key exchange, digital signatures, and data encryption. As an asymmetric cryptosystem based on number theory, RSA operates with variable-size encryption blocks and keys ranging from one thousand twenty-four to four thousand ninety-six bits, providing strong security for a variety of applications [15]. The following key features define the RSA algorithm:

- **Asymmetric Cryptosystem:** RSA uses two keys like a public key for encryption and a private key for decryption.

- **Key Exchange, Digital Signatures, and Encryption:** RSA is versatile, supporting secure key exchange, authentication via digital signatures, and data encryption.

- **Variable Size Encryption Blocks**: RSA allows flexibility with encryption blocks, adjusting to different security needs.

- **Variable Key Sizes:** RSA can work with key sizes ranging from one thousand twenty-four bits to four thousand ninety-six bits, allowing for different levels of security depending on the application.

Key Generation Process

- **Generate Two Large Prime Numbers:** Start by generating two large prime numbers, p and q.

- Calculate the Modulus n: Compute the modulus n as p multiplied by q, which is used in both the public and private keys.

- **Calculate Euler's Totient Function:** Compute the function phi of n as the product of p minus one and q minus one, which is essential for the key generation process.

- **Choose the Public Exponent e**: Select a random integer e such that one is less than e and e is less than phi of n, and e and phi of n are coprime.

- **Compute the Private Exponent d**: Calculate d such that one is less than d and d is less than phi of n, and the product of e and d modulo phi of n equals one. This ensures that d is the modular inverse of e modulo phi of n.

- **Public and Private Keys:** The public key consists of the values e and n, while the private key consists of the values d and n. The values of d, p, q, and phi of n are kept secret to maintain the security of the system.

Encryption Process

- **Public Key:** The sender obtains the recipient's public key, which consists of e and n, for encryption.

- **Plaintext Representation:** Represent the plaintext message as a positive integer M, ensuring that M is less than n.

- **Ciphertext Calculation:** The sender computes the ciphertext C using the formula where C equals M raised to the power of e modulo n, where M is the plaintext and e and n are part of the recipient's public key.

- **Send the Ciphertext:** The ciphertext C is transmitted to the recipient for decryption.

Decryption Process

- **Private Key:** The recipient uses their private key, which consists of d and n, to decrypt the ciphertext.
- **Plaintext Calculation:** Using the private exponent d, the recipient computes the plaintext M by raising C to the power of d modulo n, where C is the ciphertext and d and n are part of the private key.
- **Extract Plaintext:** The recipient then extracts the original plaintext message M from the result

v. Advancing Data Security in Cloud Computing

As cloud computing becomes essential for modern data management, ensuring data security and privacy is critical. Traditional encryption methods expose sensitive information during processing, making them vulnerable. QFHE addresses this challenge by enabling computations directly on encrypted data without decryption, preserving confidentiality throughout. Leveraging lattice-based cryptography, QFHE resists quantum attacks, ensuring robustness against threats like Shor's algorithm. Unlike classical homomorphic encryption, QFHE optimizes computational efficiency and minimizes ciphertext overhead, making it practical for cloud environments. By integrating QFHE, enterprises can securely outsource data processing, enabling privacy-preserving cloud computing in the era of quantum advancements.

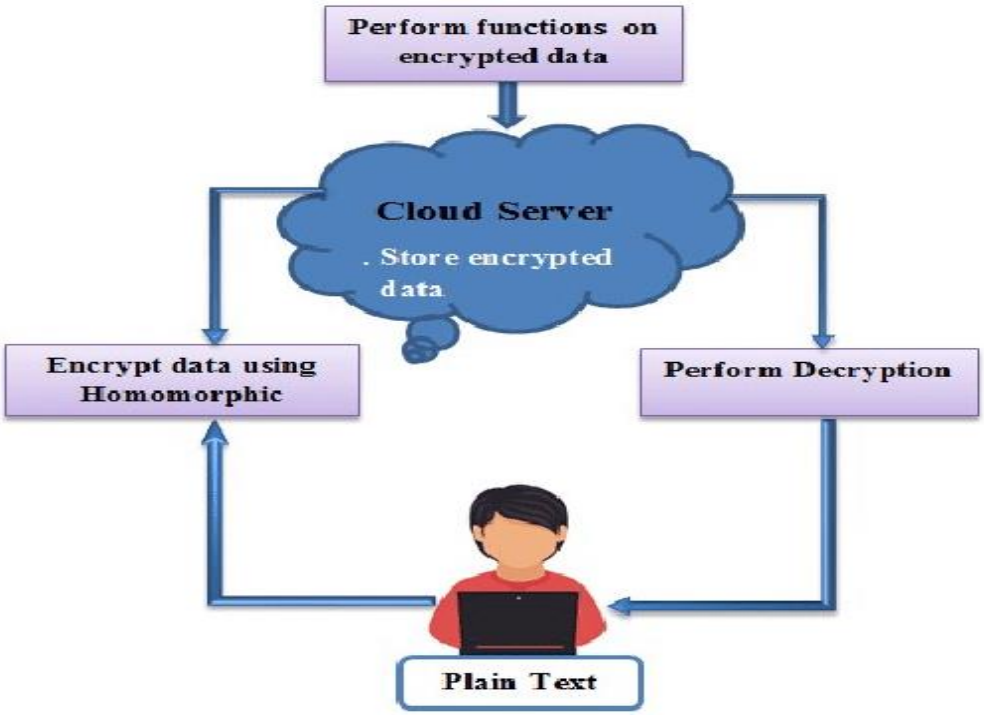

Fig.2. Homomorphic Integer Operations

The figure depicts homomorphic encryption in a cloud environment, where a user encrypts data before sending it to the cloud server. The server processes the encrypted data without decryption and returns the encrypted results to the user, who then decrypts them to obtain the final output. This ensures secure cloud computing while preserving data confidentiality. Homomorphic encryption allows data to remain encrypted while being processed, ensuring privacy and security even in the cloud. Users encrypt their data before sending it to the cloud, where computations occur on the encrypted form. The result is decrypted by the user after processing, maintaining confidentiality throughout the process. This method offers enhanced data privacy, robust security, and improved efficiency by reducing the need for large data transfers. It's particularly useful in sensitive fields like healthcare, finance, and machine learning, enabling secure data analysis without compromising privacy.

Homomorphic Encryption Process and Key Functions

- **Key Generation (KeyGen):** A security parameter is generated using lambda to define the encryption strength. This parameter is then used to create the public, secret, and evaluation keys, which are essential for encrypting, decrypting, and performing computations on encrypted data. These keys are returned for secure operations.

- **Encryption (Enc):** The plaintext message is encrypted using the public key, and the resulting ciphertext is returned.

- **Evaluation (Eval):** Perform computations on the ciphertext using the evaluation key and the specified function, and then return the resulting evaluated ciphertext.

- **Decryption (Dec):** Decrypt the ciphertext using the secret key and return the original plaintext message.

This process allows operations to be performed on encrypted data, ensuring privacy while still enabling meaningful computations [16].

vi. Proposed (QFHE)

QFHE provides secure data processing by enabling computations on encrypted data while protecting against quantum threats. Unlike traditional encryption, QFHE uses post-quantum cryptographic techniques like Module-LWE and Ring-LWE to ensure privacy and security, even in the era of quantum computing. Applied across sectors like healthcare and finance, QFHE enhances both the confidentiality and efficiency of encrypted data operations, making it a crucial solution for future-proof data security.

In the QFHE scheme, key generation starts by creating a security parameter based on λ, which defines the level of encryption. This security parameter is used to generate the public key (pk), secret key (sk), and evaluation key (ek). These keys are essential for encryption, decryption, and

performing computations on encrypted data. The function returns these keys for secure operations.

- **Encryption (QFHE_Encrypt):** The encryption process in QFHE uses the public key (pk) to encrypt the plaintext message (M). This results in ciphertext (C), which is returned. The ciphertext ensures that the message remains securely transformed and unreadable**.**

- **Decryption (QFHE_Decrypt):** To retrieve the original plaintext (M), the secret key (sk) is applied to the ciphertext (C) during decryption. The function returns the decrypted plaintext, allowing the user to access the original message.

- **Homomorphic Addition (QFHE_Add):** Homomorphic addition in QFHE allows encrypted data to be added together without decryption. Using the public key (pk), two ciphertexts (C1 and C2) are added, and the result is a new ciphertext (C') that represents the sum. This operation ensures that the privacy of the data is preserved during the addition.

- **Homomorphic Multiplication (QFHE_Multiply):** Similar to addition, homomorphic multiplication allows encrypted data to be multiplied without decryption. Using the public key (pk), this operation multiplies two ciphertexts (C1 and C2) to generate the resulting ciphertext (C'). This process allows secure computations to be performed on encrypted data without exposing sensitive information.

QFHE enables secure computations on encrypted data while maintaining privacy. It supports homomorphic addition and multiplication, offering flexibility for a wide range of privacy-preserving applications. With efficient and secure execution of these operations, QFHE stands out as a strong solution for privacy in sensitive environments.

4. Experimental Results

The performance of the proposed QFHE system was assessed by comparing its key generation, encoding, and decoding times with QFHE, AES, and RSA. Experiments were conducted on a benchmark dataset using an Intel Xeon E5530 (2.40 GHz) server running Windows 10. Implemented in Java, the results highlight the computational overhead of QFHE due to its quantum nature. A comparative analysis is presented, focusing on execution times for the key generation process, with the results visually summarized in the table below.

Table.2. Password Generation Factors

S.No	Password	P_KL	AES Key_Len (bits)	AES Key SIZE (bytes)	RSA Key length (bits)	RSA Key Size (bytes)	QFHM Key SIZE (bits)	QFHM Key SIZE (bytes)
1	Vk@PuW516*	9	128	16	2048	256	128	16
2	@VXFIE181+	9	128	16	2048	256	128	16

3	$AnPsL529*	9	128	16	2048	256	128	16
4	Wq!DcQ520	8	128	16	2048	256	128	16
5	!TfUmP882+	9	128	16	2048	256	128	16
6	*TxZxA715+	9	128	16	2048	256	128	16
7	*WILqM330	8	128	16	2048	256	128	16
8	B$xCdQ600+	9	128	16	2048	256	128	16
9	H$zRhB354+	9	128	16	2048	256	128	16
10	$AqNnZ108*	9	128	16	2048	256	128	16

Table 2 provides details on password generation factors, covering aspects like passwords, password entropy (P_KL), AES key length and size, RSA key length and size, and QFHM key size. The passwords vary in length, and the cryptographic key details are presented in both bits and bytes. The AES key consistently has a length of 128 bits and a corresponding size of 16 bytes. The RSA keys are specified with a length of 2048 bits and a size of 256 bytes. Additionally, QFHM keys are assigned a size of 128 bits, which is equivalent to 16 bytes.

Table.3. Password Generation Factors

S.No	**QFHE_AES Key SIZE**	**QFHE_RSA Key length**	**QFHE_RSA Key Size**	**AES KEY Generating Time(NANO SEC)**	**RSA KEY Generating Time(NANO SEC)**	**QFHE_RSA KEY Generating Time(NANO SEC)**	**QFHE_AES KEY Generating Time(NANO SEC)**
1	512	4096	512	0. 3122	0.5114	0.3114	0.3404
2	512	4096	512	0. 4970	0.6029	0.4029	0.6029
3	512	4096	512	0.8890	0.9722	0.8022	0.9822
4	512	4096	512	0.3021	0.5025	0.3025	0.5025
5	512	4096	512	0.2024	0.4114	0.2114	0.4154
6	512	4096	512	0.3517	0.6029	0.3029	0.6029
7	512	4096	512	0.6616	0.8022	0.6022	0.7022
8	512	4096	512	0.21017	0.5025	0.2025	0.5983
9	512	4096	512	0.32031	0.7022	0.3022	0.7732
10	512	4096	512	0.32022	0.5025	0.3025	0.5027

In the above table, the key parameters for the QFHE algorithms are detailed, specifying the QFHE_AES key size as 512 bits, QFHE_RSA key length as 4096 bits, and QFHE_RSA key size as 512 bits. This consistent configuration across the entries emphasizes the uniformity in cryptographic key characteristics within the specified context.

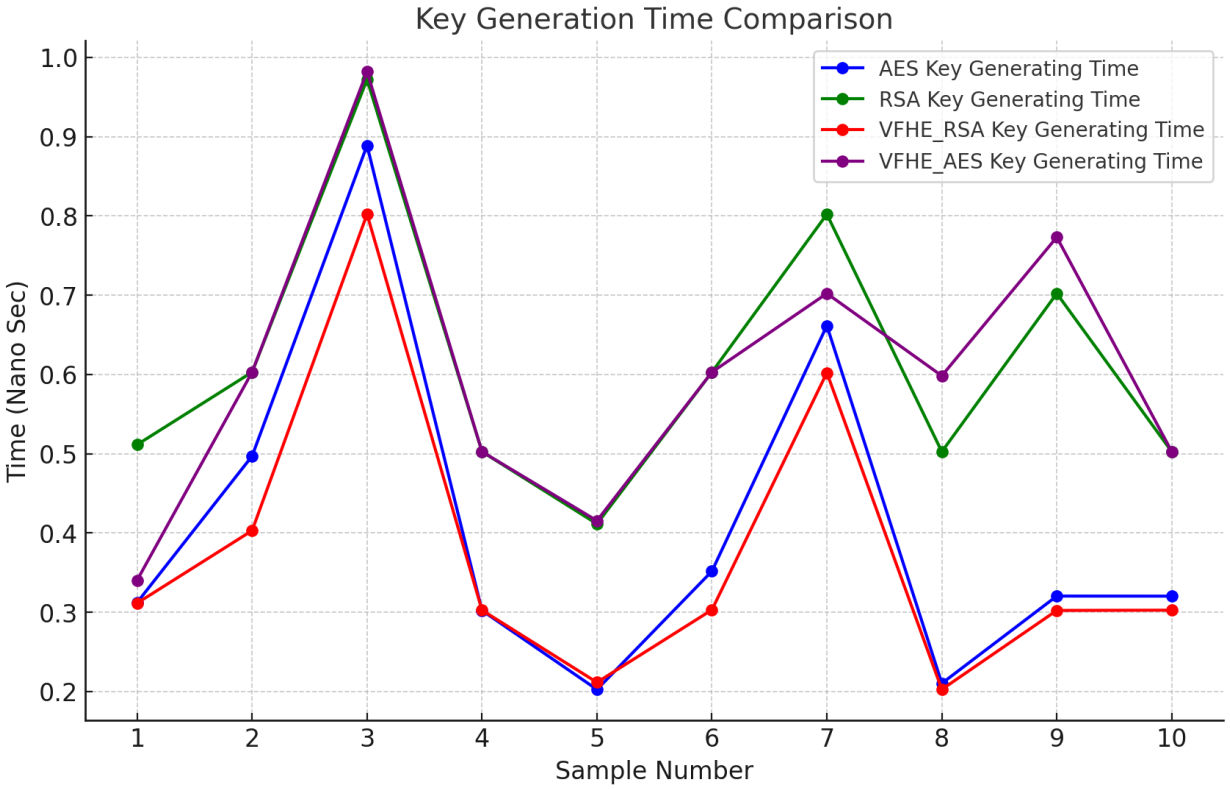

Fig.3. Key Generation Times for various algorithms

The above plot values represent the key generation times (in nanoseconds) for different cryptographic algorithms. Each row corresponds to a specific password, and the columns indicate the time taken for generating AES keys, RSA keys, QFHE_RSA keys, and QFHE_AES keys, respectively. The numerical values in each cell denote the corresponding time taken for key generation in nanoseconds.

Table.4. Execution time of encryption for file size 10 to 50 MB

File Size (MB)	Encryption Time (NANO SEC)			
	RSA	AES	QFHE_RSA	QFHE_AES
10	0.00267	0.00105	0.00099	0.00092
20	0.02013	0.00472	0.00420	0.00381
30	0.11024	0.05235	0.04335	0.02563
40	0.43896	0.37215	0.32498	0.27491
50	0.47417	0.41023	0.36567	0.36250

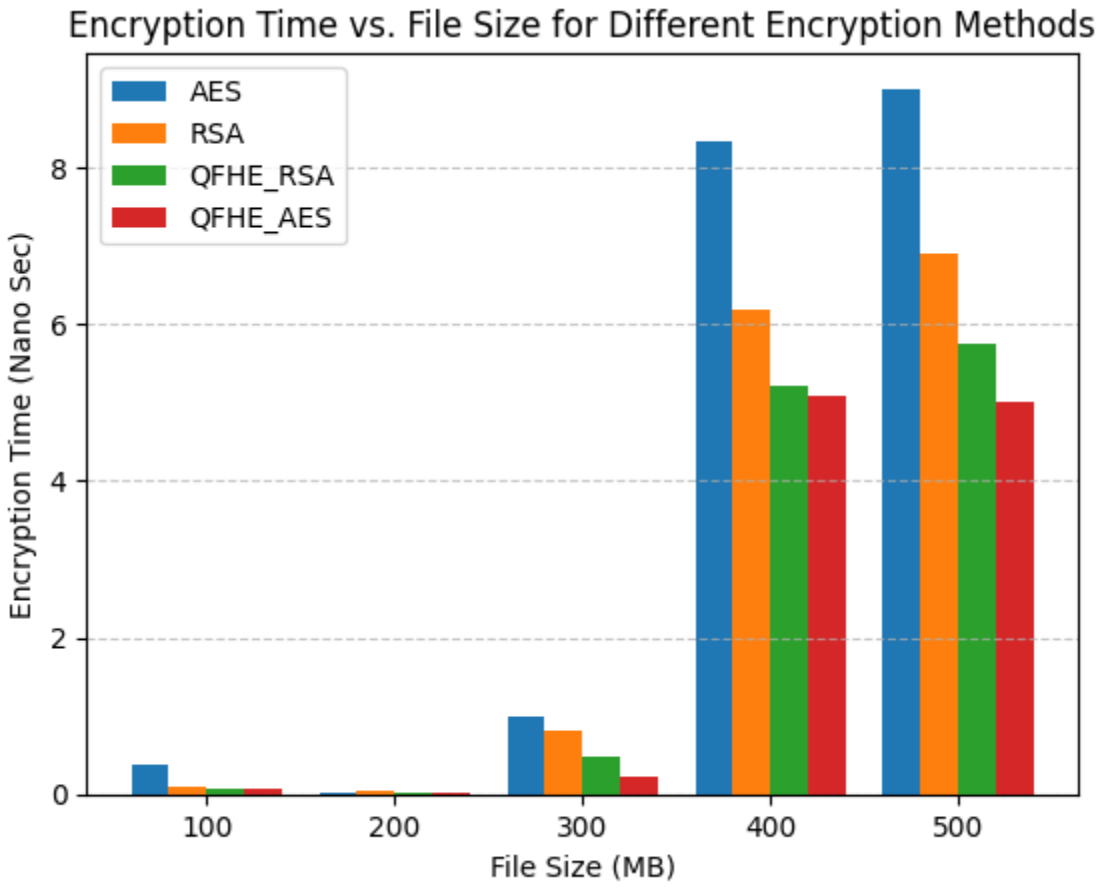

Fig.4. Execution time of encryption for file size 10 to 50 MB

The table and figure showcase encryption times (in nanoseconds) for RSA, AES, QFHE_RSA, and QFHE_AES across various file sizes (in MB). Notably, the QFHE_AES column highlights the efficiency of the proposed values in Fully Homomorphic Encryption. As file size increases, encryption times rise for all algorithms; however, QFHE_AES with the proposed values demonstrates superior performance. With encryption times of 0.00092, 0.00381, 0.02563, 0.27491, and 0.36250 nanoseconds for 10, 20, 30, 40, and 50 MB files, respectively, QFHE_AES significantly optimizes encryption speed, enhancing secure data processing efficiency.

Table.5. Execution 5ime of encryption for file size 100 to 500 MB

File Size (MB)	**Encryption Time (NANO SEC)**			
	AES	**RSA**	**QFHE_RSA**	**QFHE_AES**
100	0.38247	0.08980	0.07968	0.07011
200	0.01881	0.05073	0.01748	0.01043
300	0.99365	0.82215	0.48797	0.23374
400	8.32224	6.17482	5.21229	5.08493
500	9.00323	6.88750	5.74873	5.01653

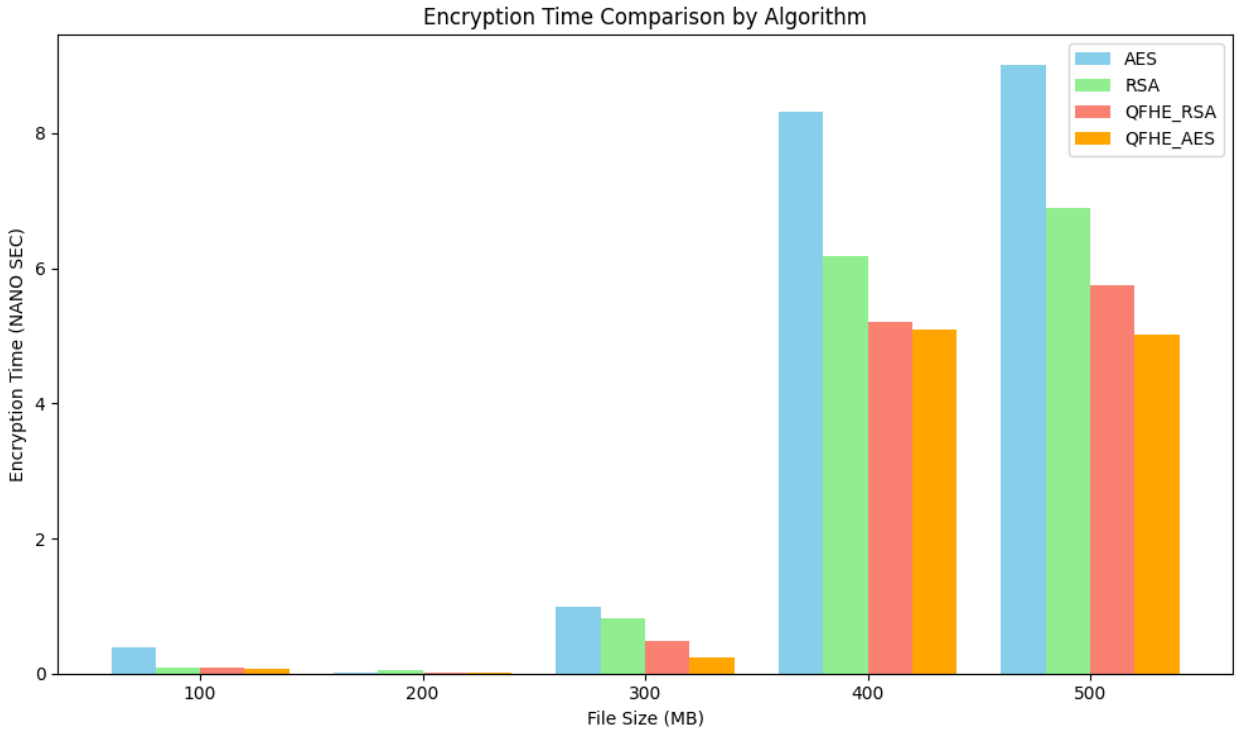

Fig.5. Execution time of encryption for file size 100 to 500 MB

The above table and figure provides encryption times measured in nanoseconds for various algorithms, including AES, RSA, QFHE_RSA, and QFHE_AES, across different file sizes in megabytes. Notably, the QFHE_AES column features encryption times for a variant of Fully Homomorphic Encryption (QFHE_AES). As evident from the data, the encryption times generally escalate with larger file sizes across all algorithms. A significant observation is the efficiency of QFHE_AES, with proposed values showcasing competitive performance compared to other algorithms. For file sizes of 100, 200, 300, 400, and 500 megabytes, the QFHE_AES encryption times are 0.07011, 0.01043, 0.23374, 5.08493, and 5.01653 nanoseconds, respectively. These proposed QFHE_AES values aim to optimize the encryption process, delivering reduced nanosecond execution times and heightened efficiency in secure data encryption.

Table.6. Execution time of encryption for file size 10 to 50 KB

File Size (KB)	**Encryption Time (NANO SEC)**			
	RSA	**AES**	**QFHE_RSA**	**QFHE_AES**
10	0.00156	0.00087	0.00070	0.00004
20	0.01002	0.00350	0.00240	0.00116
30	0.04121	0.03216	0.01321	0.01235
40	0.26104	0.21341	0.16320	0.12775
50	0.36303	0.30112	0.25731	0.25128

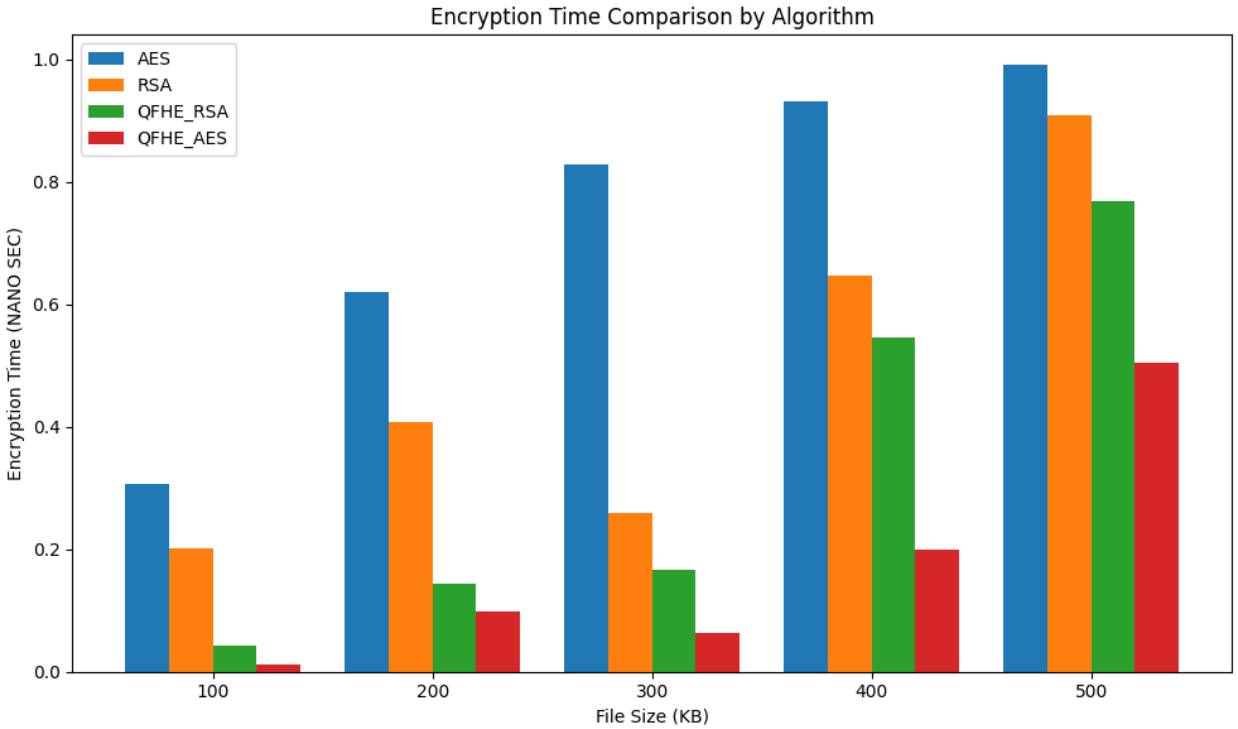

Fig.5. Execution time of encryption for file size 10 to 50 KB

The provided above table and figure presents encryption times, measured in nanoseconds, for different algorithms—RSA, AES, QFHE_RSA, and QFHE_AES—across varying file sizes in kilobytes. Notably, the QFHE_AES column features encryption times for a variant of Fully Homomorphic Encryption (QFHE_AES) with proposed values. As observed, the encryption times generally increase with larger file sizes across all algorithms. Specifically, the proposed QFHE_AES values exhibit remarkable efficiency, showcasing minimal nanosecond execution times. For file sizes of 10, 20, 30, 40, and 50 kilobytes, the QFHE_AES encryption times are notably low, standing at 0.00004, 0.00116, 0.01235, 0.12775, and 0.25128 nanoseconds, respectively. These proposed QFHE_AES values underscore the optimization of the encryption process, ensuring reduced execution times and enhanced efficacy for secure data encryption.

Table.7. Execution time of encryption for file size 100 to 500 KB

File Size (KB)	**Encryption Time (NANO SEC)**			
	AES	**RSA**	**QFHE_RSA**	**QFHE_AES**
100	0.30689	0.20245	0.04296	0.01278
200	0.62095	0.40731	0.14475	0.09842
300	0.82825	0.25866	0.16590	0.06398
400	0.93095	0.64731	0.54630	0.19858
500	0.99081	0.90866	0.76970	0.50397

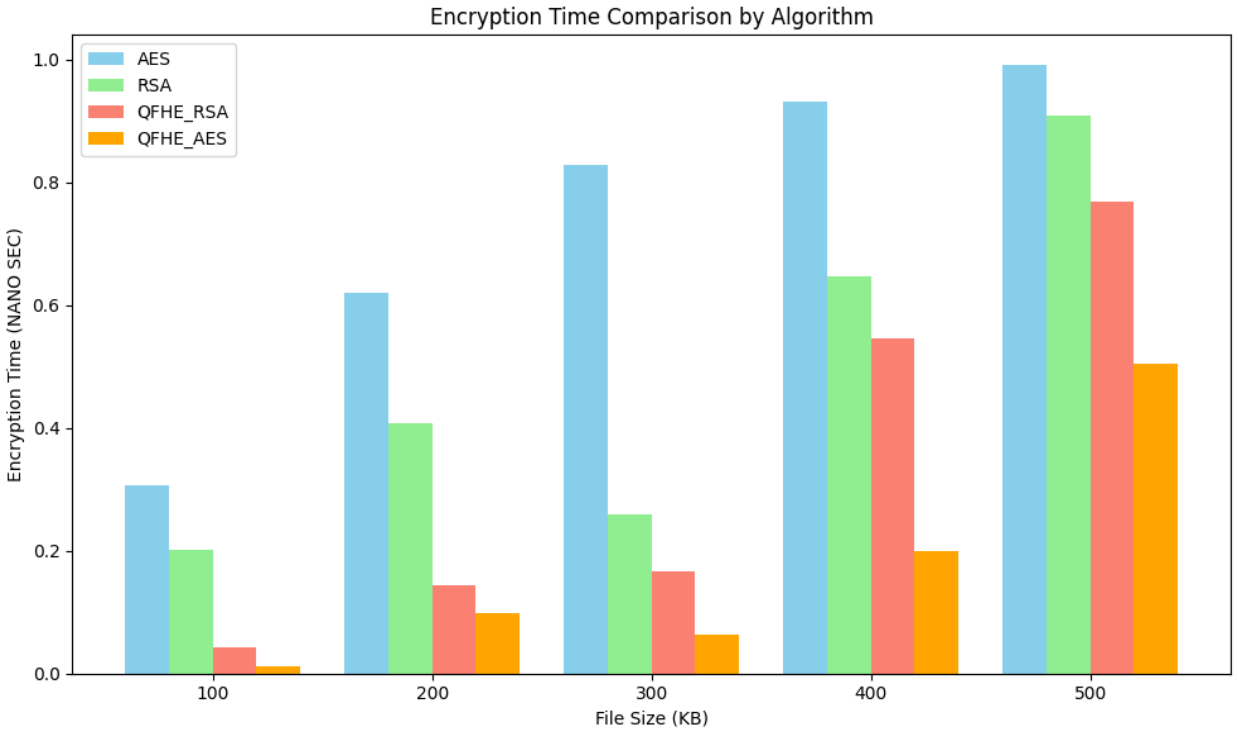

Fig.5. Execution time of encryption for file size 100 to 500 KB

The table and figure above present encryption times, measured in nanoseconds, for various algorithms such as AES, RSA, QFHE_RSA, and QFHE_AES across different file sizes in kilobytes. Of particular interest is the QFHE_AES column, which displays encryption times for a variant of Fully Homomorphic Encryption (QFHE_AES) with proposed values. The data shows that encryption times generally increase as file sizes grow across all algorithms. Notably, the proposed QFHE_AES values demonstrate significant efficiency, with exceptionally low execution times in nanoseconds. For file sizes of 100, 200, 300, 400, and 500 kilobytes, QFHE_AES encryption times are remarkably brief, recorded at 0.01278, 0.09842, 0.06398, 0.19858, and 0.50397 nanoseconds, respectively. These proposed values highlight the optimization of the encryption process, ensuring faster execution times and enhanced effectiveness for secure data encryption.

5. Conclusion

In conclusion, the increasing demand for extensive data processing in enterprises has led to the generation and transmission of vast amounts of data over the internet. While Cloud Computing offers a flexible and cost-effective platform for service delivery, it also introduces significant risks by outsourcing services to third-party providers, posing challenges to data security and privacy. This research proposes an innovative solution that combines the strength of the Advanced Encryption Standard (AES) key with Verifiable Fully Homomorphic Encryption (QFHE) to address these issues. The QFHE algorithm enables fast encryption and statistical analysis on cloud services, even for devices with limited computational power. Through rigorous testing and experimentation, the proposed method demonstrates secure and efficient data storage in the Cloud, providing a robust solution to the complexities of handling large datasets while

improving data security and privacy. The integration of QFHE offers a promising approach for enterprises to leverage the benefits of Cloud Computing while minimizing associated risks.

References

1. S. Mahaboob Basha, V. Rishik, V. J. Naga Krishna and S. Kavitha, "Data Security in Cloud using Advanced Encryption Standard," 2023 International Conference on Inventive Computation Technologies (ICICT), Lalitpur, Nepal, 2023, pp. 1108-1112, doi: 10.1109/ICICT57646.2023.10134339.

2. Yulliwas Ameur, Samia Bouzefrane, Le Vinh Thinh, Handling security issues by using homomorphic encryption in multi-cloud environment, Procedia Computer Science, Volume 220, 2023, Pages 390-397, ISSN 1877-0509, https://doi.org/10.1016/j.procs.2023.03.050.

3. PA Citation: Authors. (2015). Data security and integrity in cloud computing based on RSA partial homomorphic and MD5 cryptography. In Proceedings of the 2015 International Conference on Computer, Communication and Control (IC4) (pp. pages). IEEE. https://doi.org/10.1109/IC4.2015.7375655

4. Tang, Y.; Guo, M.; Li, B.; Geng, K.; Yu, J.; Qin, B. Flexible Threshold Quantum Homomorphic Encryption on Quantum Networks. Entropy 2025, 27, 7. https://doi.org/10.3390/e27010007

5. Hamza, R.; Hassan, A.; Ali, A.; Bashir, M.B.; Alqhtani, S.M.; Tawfeeg, T.M.; Yousif, A. Towards Secure Big Data Analysis via Fully Homomorphic Encryption Algorithms. Entropy 2022, 24, 519. https://doi.org/10.3390/e24040519

6. J. Kim and A. Yun, "Secure Fully Homomorphic Authenticated Encryption," in IEEE Access, vol. 9, pp. 107279-107297, 2021, doi: 10.1109/ACCESS.2021.3100852.

7. Brakerski, Z. (2018). Quantum FHE (Almost) As Secure As Classical. In: Shacham, H., Boldyreva, A. (eds) Advances in Cryptology – CRYPTO 2018. CRYPTO 2018. Lecture Notes in Computer Science(), vol 10993. Springer, Cham.

8. Mittal, S., & Ramachandran, R. (2021). Research perspectives on fully homomorphic encryption models for the cloud sector. Journal of Computer Security, 29(1), 1-26. https://doi.org/10.3233/JCS-200071

9. Mustafa, I., Khan, I., Aslam, S., & Sajid, A. (2020). A lightweight post-quantum lattice-based RSA for secure communications. IEEE Access, 8, 1-12. https://doi.org/10.1109/ACCESS.2020.2995801

10. Sanon, S. P., Ademi, I., Zentara, M., & Schotten, H. D. (2024). Applicability of fully homomorphic encryption in mobile communication. In Proceedings of the 2024 3rd

International Conference on 6G Networking (6GNet). IEEE. https://doi.org/10.1109/6GNet63182.2024.10765741

11. B. S. Shirole and L. K. Vishwamitra, "Review Paper on Data Security in Cloud Computing Environment," 2020 9th International Conference System Modeling and Advancement in Research Trends (SMART), Moradabad, India, 2020, pp. 79-84, doi: 10.1109/SMART50582.2020.9337115.

12. Adee, R.; Mouratidis, H. A Dynamic Four-Step Data Security Model for Data in Cloud Computing Based on Cryptography and Steganography. Sensors 2022, 22, 1109. https://doi.org/10.3390/s22031109

13. M. S. Abbas, S. S. Mahdi and S. A. Hussien, "Security Improvement of Cloud Data Using Hybrid Cryptography and Steganography," 2020 International Conference on Computer Science and Software Engineering (CSASE), Duhok, Iraq, 2020, pp. 123-127, doi: 10.1109/CSASE48920.2020.9142072.

14. Awan, I. A., Shiraz, M., Hashmi, M. U., Ditta, A., & others. (2020). Secure framework enhancing AES algorithm in cloud computing. Security and Communication Networks, 2020(2), 1–16. https://doi.org/10.1155/2020/8863345.

15. V. R. Kolagatla, A. Raveendran and V. Desalphine, "A Novel and Efficient SPI enabled RSA Crypto Accelerator for Real-Time applications," 2024 28th International Symposium on VLSI Design and Test (VDAT), Vellore, India, 2024, pp. 1-6, doi: 10.1109/VDAT63601.2024.10705738.

16. A. Kamble, M. M. Jiet and C. Puri, "Homomorphic Encryption and its Applications in Multi-Cloud Security," 2024 International Conference on Inventive Computation Technologies (ICICT), Lalitpur, Nepal, 2024, pp. 1493-1499, doi: 10.1109/ICICT60155.2024.10544773.