Article

Secure and Scalable File Encryption for Cloud Systems via Distributed Integration of Quantum and Classical Cryptography

Changjong Kim [1], Seunghwan Kim [1], Kiwook Sohn [1], Yongseok Son [2], Manish Kumar [3] and Sunggon Kim [1,*]

[1] Department of Computer Science, Seoul National University of Science and Technology, Seoul 01811, Republic of Korea; changjong5238@seoultech.ac.kr (C.K.); rlatmdghkss@seoultech.ac.kr (S.K.); kiwook@seoultech.ac.kr (K.S.)
[2] Department of Computer Science and Engineering, Chung-Ang University, Seoul 06974, Republic of Korea; sysganda@cau.ac.kr
[3] Department of Computer Engineering (AI-ML), Marwadi University, Rajkot 360003, India; manish.guptasssss007@gmail.com
* Correspondence: sunggonkim@seoultech.ac.kr

Abstract

We propose a secure and scalable file-encryption scheme for cloud systems by integrating Post-Quantum Cryptography (PQC), Quantum Key Distribution (QKD), and Advanced Encryption Standard (AES) within a distributed architecture. While prior studies have primarily focused on secure key exchange or authentication protocols (e.g., layered PQC-QKD key distribution), our scheme extends beyond key management by implementing a distributed encryption architecture that protects large-scale files through integrated PQC, QKD, and AES. To support high-throughput encryption, our proposed scheme partitions the target file into fixed-size subsets and distributes them across slave nodes, each performing parallel AES encryption using a locally reconstructed key from a PQC ciphertext. Each slave node receives a PQC ciphertext that encapsulates the AES key, along with a PQC secret key masked using QKD based on the BB84 protocol, both of which are centrally generated and managed by the master node for secure coordination. In addition, an encryption and transmission pipeline is designed to overlap I/O, encryption, and communication, thereby reducing idle time and improving resource utilization. The master node performs centralized decryption by collecting encrypted subsets, recovering the AES key, and executing decryption in parallel. Our evaluation using a real-world medical dataset shows that the proposed scheme achieves up to 2.37× speedup in end-to-end runtime and up to 8.11× speedup in encryption time over AES (Original). In addition to performance gains, our proposed scheme maintains low communication cost, stable CPU utilization across distributed nodes, and negligible overhead from quantum key management.

Keywords: Post-Quantum Cryptography; Quantum Key Distribution; Advanced Encryption Standard; cloud system

check for updates

Academic Editors: Georgios M. Nikolopoulos and Alexander Barkalov

Received: 20 May 2025
Revised: 27 June 2025
Accepted: 7 July 2025
Published: 11 July 2025

Citation: Kim, C.; Kim, S.; Sohn, K.; Son, Y.; Kumar, M.; Kim, S. Secure and Scalable File Encryption for Cloud Systems via Distributed Integration of Quantum and Classical Cryptography. *Appl. Sci.* **2025**, *15*, 7782. https://doi.org/10.3390/app15147782

1. Introduction

The volume and diversity of data have been growing rapidly across domains such as AI, healthcare, finance, and scientific simulation [1–5]. Traditional data are typically structured and well defined, such as relational tables and transaction logs. In contrast, modern data are dynamic, unstructured, and continuously generated from diverse sources, including real-time sensor streams, financial transactions, and user interactions. These data streams contain sensitive information, including patient vitals, account credentials, and behavioral patterns [6–9].

Additionally, unstructured data continue to grow rapidly due to real-time generation and continuous accumulation in data-intensive applications [10,11]. This continued growth in volume and sensitivity has made the ability to securely process such large-scale files a critical system requirement.

As data become increasingly large-scale, complex, and security-critical, computing infrastructures have adopted distributed and heterogeneous architectures. Cloud systems such as AWS, Google Cloud, and Azure [12–14] support scalability and high availability, while also providing robust security environments. These platforms implement protocols like TLS [15] for secure transmission and Identity and Access Management (IAM) [16] for account control.

The emergence of quantum computing, however, introduces new security threats, as these infrastructures typically rely on classical encryption methods to protect sensitive data [17–19]. Quantum computers offer high computational efficiency but undermine core assumptions of existing classical cryptographic algorithms. For example, Shor's algorithm [20] breaks RSA and ECC by solving prime factorization and discrete logarithms, while Grover's algorithm [21] accelerates brute-force key searches against symmetric schemes like AES [22]. Although AES remains relatively secure under quantum attacks with sufficient key length (e.g., AES-256), its security depends on the assumption that the symmetric key is established securely. However, since RSA or ECC is commonly used to exchange AES session keys, the entire encryption becomes vulnerable if the key-exchange mechanism is broken by quantum adversaries. To address this challenge, encryption schemes should be designed to provide security and compatibility in both classical and quantum computing environments. Among available solutions, Post-Quantum Cryptography (PQC) [23] and Quantum Key Distribution (QKD) [24] are the most widely utilized techniques for quantum-resilient security. PQC relies on hard mathematical problems to resist quantum attacks [25–27], while QKD uses quantum mechanics to distribute keys securely [28–31].

However, both PQC and QKD face limitations when used independently. PQC can securely encapsulate AES keys for data encryption, but lacks mechanisms to verify the integrity of the encapsulated key or prevent tampering. In distributed systems where the same key is broadcast to multiple nodes, PQC without additional authentication cannot ensure the origin or integrity of the transmitted key, making it vulnerable to man-in-the-middle (MITM) attacks and ciphertext forgery [32–34]. In contrast, QKD enables secure key exchange using quantum channels but offers no way to verify the identity of the sender. Without authentication support, a receiver cannot confirm the origin of the key, leaving the system open to impersonation or interception. QKD is fundamentally designed for key distribution and lacks mechanisms to store, format, or directly apply keys in encryption workflows [28,35,36]. For practical data protection, it should be integrated with a symmetric encryption method.

To overcome these limitations, recent studies have actively explored combining these techniques to strengthen cryptographic systems. Table 1 summarizes recent approaches integrating quantum and classical cryptography. Specifically, Zeng et al. [37] proposed a hybrid key-distribution protocol combining PQC and QKD to ensure security even if one scheme fails, focusing on theoretical key security models and vulnerability metrics. Yang et al. [38] designed a QKD-enabled authentication framework for Internet of Vehicles by integrating BB84-based key exchange with lattice-based digital signatures, emphasizing mutual authentication and resistance to quantum attacks. Zeydan et al. [39] presented a blockchain-based service-orchestration platform that secures network-management logs using PQC-enhanced digital signatures and evaluates the performance of Blockchain Network (BCN) operations under quantum-safe computations. Ricci et al. [40] proposed a concrete FPGA implementation of a 3-key combiner that integrates keys from pre-quantum, post-quantum, and QKD sources using a dual-PRF construction. Wang et al. [41] experimentally verified using a PQC-based digital signature to authenticate the classical channel in a QKD network, solving the key-management problem for large-scale networks. Rani et al. [42] proposed a prototype QKD system that employs sequential encryption, first with a QKD key and then with AES-256, to ensure security even if one of the primitives fails. Although it does not explicitly adopt PQC algorithms, AES-256 is used as a quantum-resilient encryption method, providing post-quantum level security against quantum attacks.

Table 1. Comparison of recent cryptographic approaches integrating quantum and classical techniques.

Paper	Crypto Combination	Target Scope
Zeng et al. [37]	PQC, QKD	Key Exchange
Yang et al. [38]	QKD, Digital Signature	Key Exchange
Zeydan et al. [39]	PQC, Digital Signature	Blockchain-based Security
Ricci et al. [40]	PQC, QKD	Key Exchange
Wang et al. [41]	PQC, QKD	Key Exchange
Rani et al. [42]	QKD, AES	File Transmission
Our Study	PQC, QKD, AES	File Encryption

Our study distinguishes itself from previous studies by combining classical and quantum cryptographic techniques into a unified, distributed file encryption. While earlier approaches have primarily focused on secure key exchange or communication protocols, we implement a practical encryption scheme that integrates PQC, QKD, and AES to protect large-scale files. Our proposed scheme decouples key management from encryption by assigning centralized PQC-based key encapsulation and QKD-based key masking to the master node, while enabling parallel AES encryption at slave nodes through an offset-aware partitioning method. Overlapped encryption and transmission pipelines reduce latency, and parallel decryption and reassembly at the master improve scalability and throughput across the cluster. Additionally, our scheme addresses the growing demand for secure and efficient encryption of large-scale files in distributed cloud systems, where conventional methods struggle with scalability and quantum resilience. By decoupling key management from data encryption and leveraging parallel processing, it ensures that large-scale files can be securely handled without compromising performance or long-term security.

In this paper, we propose a secure, scalable, and quantum-resilient encryption scheme designed for large-scale file protection in a cluster. The goal of our scheme is (1) to enable efficient encryption of the target file through offset-aware partitioning and parallel processing, (2) to ensure long-term security by integrating both PQC and QKD, and (3) to reduce latency and system overhead by overlapping encryption and transmission. To achieve this, the target file is divided into fixed-size subsets and distributed to slave nodes, where each subset is independently encrypted using an AES key that has been encapsulated via Kyber-based PQC. The encapsulated PQC secret key is then masked using a QKD-derived key transmitted. The encrypted subsets are asynchronously returned to the master node, where decryption is performed by decapsulating the AES key and unmasking it using the QKD key. Our evaluation results using real-world datasets confirm that, even with the added complexity of multi-layered key protection combining PQC and QKD, our scheme outperforms conventional AES by up to 8.11× in encryption speed. This highlights that strong quantum-resilient security can be achieved without sacrificing performance, ensuring efficient and secure processing across both classical and quantum computing environments.

Our contributions are as follows:

- We design a scalable encryption scheme for large-scale file protection, which enhances key security by integrating Kyber-based post-quantum key encapsulation with BB84-based quantum key distribution.
- We design and implement an offset-aware partitioning distributed encryption framework that performs subset-level AES encryption in parallel across multiple slave nodes, while decoupling key generation and transmission through centralized master node coordination.
- We demonstrate that our scheme achieves up to 8.11× faster encryption performance compared to conventional AES, while maintaining stable CPU utilization under real-world workloads.

2. Background

2.1. Post-Quantum Cryptography

As quantum computing advances, existing encryption methods used in classical computers, such as RSA, AES, and ECC [22,43,44], are becoming increasingly vulnerable. While these methods are effective against conventional mathematical attacks, they can be rendered ineffective by quantum algorithms that exploit entanglement and parallelism [45–48]. To address this, many industries and researchers most widely used post-quantum cryptography (PQC) algorithms to ensure secure encryption in the quantum era. PQC algorithms are based on mathematical problems that are difficult to solve even with quantum computing, such as lattice and multivariate polynomial problems. In addition, PQC can be integrated into existing computing systems, offering practical compatibility and flexibility during the transition to quantum-safe security technologies [26,49,50].

Among various PQC approaches, lattice-based cryptography is based on computationally hard problems such as the Shortest Vector Problem (SVP) and Learning With Errors (LWE), which remain intractable even for quantum computers [51,52]. These problems are defined in high-dimensional Euclidean space and are resistant to known quantum algorithms. A lattice-based encryption algorithm is used for key encapsulation to ensure secure key exchange. Kyber [53], based on the LWE problem and selected for NIST standardization, offers strong security and efficient performance. Kyber uses compact key sizes, performs structured matrix operations over polynomial rings, and supports parallel computation through simple and regular arithmetic patterns. These properties allow compatibility with classical systems and internet protocols, and enable efficient execution on manycore architectures such as modern CPUs and GPUs. Kyber is suitable for secure integration into distributed and cloud systems.

2.2. Quantum Key Distribution

Quantum Key Distribution (QKD) is a key-exchange method based on quantum mechanics [24]. It enables two parties to securely share a key by transmitting quantum states that reveal any eavesdropping through observable disturbances. The BB84 protocol [54] is most widely used, using qubits (i.e., single photons) encoded in two bases (e.g., rectilinear and diagonal). After transmission, the parties compare part of the results over a classical channel to detect interception. Unlike classical methods (e.g., Diffie-Hellman [55] or RSA [22]), QKD provides information-theoretic security against quantum attacks.

Figure 1 shows a symmetric encryption architecture integrated with QKD. This architecture is composed of four key components: a client that initiates requests, a symmetric-key encryptor that performs the encryption operation, a symmetric-key decryptor that recovers the original file, and QKD modules that handle quantum key generation and distribution. The encryptor and decryptor are connected through classical and quantum channels. The quantum channel enables the secure exchange of a session key using quantum states, while the classical channel carries the encrypted data. This ensures that the data path from the key path allows the encryption and key-exchange processes to proceed independently.

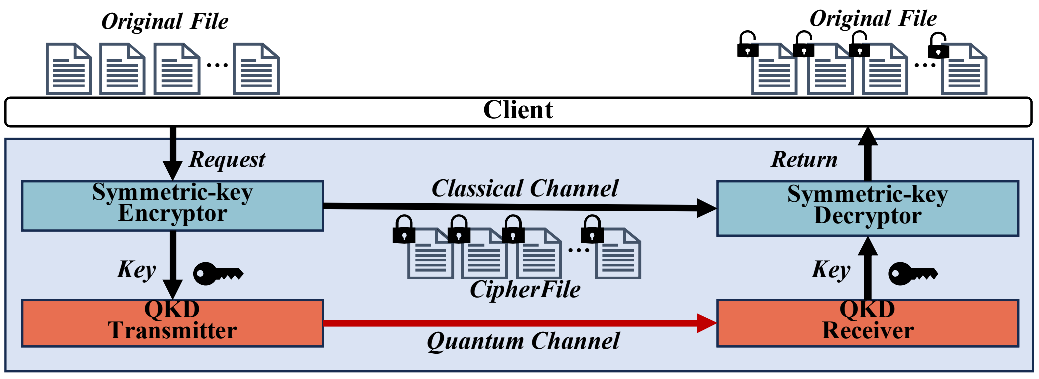


Figure 1. Quantum Key Distribution.

Specifically, as shown in the figure, the client first sends a request to initiate encryption. The symmetric-key encryptor then generates the cipher file using a symmetric encryption algorithm. Simultaneously, a quantum key is generated and transmitted from the QKD Transmitter to the QKD Receiver through a quantum channel. The generated key is used to encrypt the file, and the resulting cipher file is sent to the decryptor via a classical channel. The symmetric-key decryptor retrieves the corresponding key from the QKD Receiver and applies it to decrypt the file. Once decryption is complete, the original file (Decryption) is returned to the client. While QKD offers strong security against quantum threats, it requires specialized hardware and is noise sensitive [56–58]. To improve practicality, recent efforts aim to enhance protocol efficiency and integrate QKD into classical infrastructures [59,60].

However, in real-world cloud systems, where communication may occur over untrusted classical networks, authentication mechanisms are required to prevent active attacks. In such cases, post-quantum digital signatures such as Dilithium [61] can be integrated to provide quantum-secure authentication and ensure session integrity. Although physical QKD systems face challenges such as channel noise and hardware constraints, our proposed scheme avoids these limitations by adopting a software-emulated QKD model using Qiskit Aer 0.6.0 [62]. The software-based approach enables practical integration and reproducible evaluation, while preserving the conceptual security guarantees of QKD. Thus, our proposed scheme adopts a software-based approach that combines QKD with PQC using the Qiskit Aer simulator [62]. This layered architecture enhances the overall robustness of key management in cloud systems by integrating cryptographic techniques that address both classical and quantum threats.

2.3. Challenges in Large-Scale File Encryption

Advanced Encryption Standard (AES) [22] is widely used for file encryption due to its strong security and efficient symmetric-key structure. However, when applied to large-scale files in distributed or cloud systems, conventional AES-based implementations face several limitations. AES is typically deployed on a single node, where the entire file is loaded into memory and processed sequentially. As file sizes increase to tens or hundreds of gigabytes, this approach leads to I/O bottlenecks, excessive memory usage, and limited parallelism due to the lack of distributed processing [63–65].

However, AES is a block-based encryption scheme that supports subset-level processing through modes such as CTR [66] and CBC [67]. Even in these cases, several constraints arise, including key reuse management, initialization vector (IV) synchronization, and output ordering [68–70]. For example, in CBC mode, each block depends on the ciphertext of the previous block, making parallel processing infeasible. If the IV is not properly shared or is duplicated across nodes, ciphertext consistency will be compromised. While CTR mode enables parallel processing by treating block indices as nonces, strict key and IV scheduling is required to avoid offset collisions between nodes, which can limit parallelism. In addition, AES encryption is typically structured as a sequential process, where the tight coupling of key generation and data encryption limits parallelism in multi-node cloud systems [65,71–73]. For example, when a master node generates an AES session key and distributes it to slave nodes before encryption can begin, the need to synchronize the encryption startup across all nodes introduces initialization overhead and delays, which directly affect overall system performance. To address this, offset-aware partitioning, parallel AES processing, and centralized key coordination are essential. Our proposed scheme is designed to satisfy these challenges with scalable and secure file encryption in distributed and cloud systems.

3. Design

We propose a distributed file-encryption scheme that integrates Post-Quantum Cryptography (PQC), Quantum Key Distribution (QKD), and AES-CTR-based encryption for cloud systems composed of a master node and multiple slave nodes. The scheme enables quantum-resilient key distribution and scalable encryption performance through offset-aware partitioning and parallel encryption processing. The design is structured as a layered cryptographic stack with clearly separated roles and interfaces among PQC, QKD, and AES, which avoids redundant coverage and enables modular composition of secure components.

3.1. Overall Architecture

Figure 2 shows the overall architecture of the proposed distributed encryption scheme. Our proposed scheme consists of a single master and multiple slaves, connected through MPI-based communication.

Security model and threat assumptions: We assume a semi-honest threat model where the master node is trusted to perform key generation and distribution. Our threat model specifically considers an adversary equipped with a quantum computer, capable of breaking classical public-key algorithms such as RSA and ECC. This threat is critical because these algorithms are traditionally used to exchange symmetric keys (e.g., for AES), which could compromise the entire encryption workflow. To address this, our proposed scheme is designed to be resilient against such quantum attacks on the key-exchange mechanism through a layered integration of Kyber-based PQC for key encapsulation and QKD for protecting key transmission.

In addition, slave nodes may be partially compromised, but cannot access complete key materials due to the use of secret sharing and QKD-based key masking. All communication occurs over classical channels without built-in confidentiality. To ensure secure transmission, key materials and data subsets are encrypted before being sent. QKD operations are simulated using Qiskit Aer, and session validity is verified through QBER-based rejection. Trust in the quantum channel is limited to the eavesdropping detection properties of the BB84 protocol.

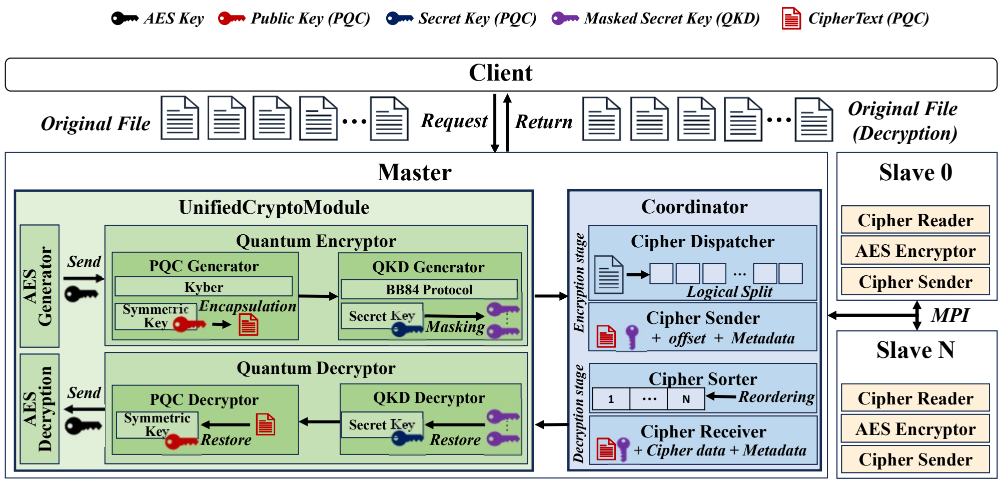


Figure 2. Overall architecture of our proposed scheme.

Master: The master is responsible for secure key generation, offset partitioning and assignment, and coordination of encryption and decryption across all slaves.

The master includes the following components: *UnifiedCryptoModule* and *Coordinator*. First of all, to improve key confidentiality, *UnifiedCryptoModule* assigns distinct roles to PQC and QKD. It generates a symmetric AES key and encapsulates it using a Kyber-based PQC key pair (e.g., public and secret keys). The encapsulated key is shared with each slave, while the PQC secret key is masked using a BB84-based QKD key. The QKD key is securely established through a BB84-based protocol between the master and each slave. To improve fault tolerance, the masked PQC secret key can optionally be divided using Shamir's secret sharing [74,75] and distributed across multiple slaves. This layered design protects different parts of the keying process against separate threats: PQC resists quantum decryption of key exchange, and QKD prevents the secret key from exposure during distribution.

Second, *Coordinator* manages both encryption and decryption stages. During encryption, *Cipher Dispatcher* logically partitions the target file into fixed-size subsets (e.g., 4, 8, or 16 MB) and assigns byte-range offsets (e.g., `offset_start`, `offset_end`) to each slave. To support parallelism, each slave independently encrypts its assigned byte range without requiring inter-node synchronization. This enables globally distributed encryption, as disjoint offsets prevent overlap and remove the need for coordination among nodes. *Cipher Sender* on the master transmits the assigned offset ranges along with encryption metadata such as the PQC ciphertext and QKD-masked secret key to each slave, enabling local AES key reconstruction and subset encryption. During decryption, *Cipher Receiver* asynchronously gathers encrypted subsets from all slaves. The collected subsets are then reordered by the *Cipher Sorter* based on subset indices (e.g., `subset_idx`). The recovered AES key is then used to decrypt all subsets in parallel using a thread pool, where each decrypted subset is immediately written to the recovered file at its designated offset. This streaming-based and offset-aware decryption approach eliminates the need to aggregate all decrypted data in memory, thereby preventing memory exhaustion and ensuring efficient and stable memory usage even when handling large files.

Slave: Each slave performs streaming encryption of its assigned file into the offset range and asynchronously returns encrypted subsets to the master. Each slave consists of the following components: *Cipher Reader*, *AES Encryptor*, and *Cipher Sender*.

Cipher Reader sequentially reads the assigned file region in units of a predefined subset size (e.g., 4, 8, or 16 MB). For each subset, *AES Encryptor* reconstructs the AES key using the received ciphertext and the QKD-unmasked secret key, then encrypts the subsets using AES in CTR mode. *Cipher Sender* batches the resulting encrypted subsets and asynchronously transmits them back to the master using MPI's non-blocking `isend()` interface. The three components operate as a pipeline, overlapping disk I/O, encryption, and communication to hide latency and maximize throughput during distributed encryption.

3.2. Layered Key Initialization and Data Partitioning

Figure 3 shows the procedure for secure key initialization and offset-aware file partitioning prior to distributed encryption. Our proposed scheme starts with generating a symmetric AES key. *AES Generator* creates a random 256-bit key [76,77] (❶). Then, *PQC Generator* performs key-pair generation using Kyber. A public key and a secret key are created. The AES key is encapsulated using the PQC public key, resulting in a PQC ciphertext. This encapsulated ciphertext enables the AES key to be securely transferred to each slave node without revealing its plaintext. The PQC ciphertext is included in the metadata sent to each slave, while the corresponding PQC secret key is securely retained at the master in protected form for later AES key recovery. To prevent leakage of the PQC secret key during transmission, *QKD Generator* runs a BB84-based protocol to generate a QKD session key. QKD session key is used to XOR-mask the PQC secret key before transmitting it. Masked secret key can be recovered only by a slave that holds the same QKD session key. Even if intercepted, the masked secret key remains secure, as it cannot be used without the corresponding QKD session key. For fault tolerance and resilience against node failure, the masked secret key is optionally split into multiple fragments using Shamir's (t, n) secret sharing scheme. These fragments are distributed to n slaves, such that any t of them can later reconstruct the original masked key (❷). Although all slaves receive a fragment and reconstruct the masked secret key to perform AES encryption, only a designated subset of t nodes is authorized to retain the reconstructed key beyond the encryption phase. The remaining slaves discard the key immediately after encryption, preventing long-term exposure even under partial compromise. This strategy enables fault-tolerant key recovery while preserving post-execution confidentiality. After key encapsulation and masking, the master sends the PQC ciphertext and the masked secret key to each slave. This is because the secret key is never exposed in plaintext and is recoverable only through QKD-protected masking. Our scheme maintains confidentiality even during key distribution. These cryptographic components are delivered before file partitioning begins, enabling each slave to focus solely on subset-level encryption without participating in key generation or coordination (❸).

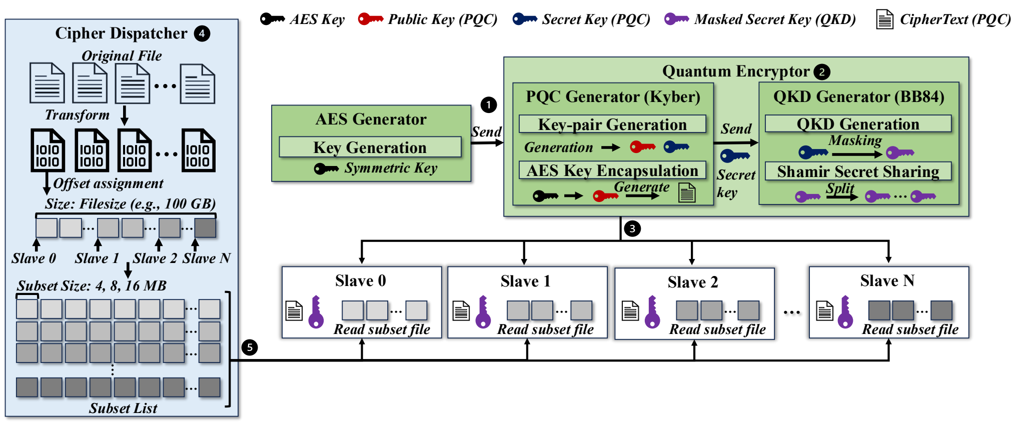


Figure 3. Layered key initialization and offset-aware partitioning for distributed encryption.

Once key transmission is complete, *Cipher Dispatcher* performs data preparation and partitioning for parallel encryption. The target file is first converted into binary, and its total size is measured. Based on the number of participating slave nodes, the master computes logical start and end offsets for each node, effectively dividing the file into disjoint byte-range regions. These regions are designed to support concurrent access by slave nodes during subset-level encryption without overlap or contention. Following offset computation, the file is conceptually divided into fixed-size subsets (e.g., 4, 8, and 16 MB), and the total number of subsets is determined from the file size and subset configuration. The resulting subset layout defines the unit of encryption and completes the preparation phase for distributed execution (❹). Once offset ranges are defined and the file is segmented into fixed-size subsets, each slave receives its designated offset information for file access. During encryption, each slave reads the file from the assigned offset range in a streaming manner, processing the data in fixed-size subsets. For each subset, the AES key is locally reconstructed by decapsulating the received PQC ciphertext using the masked secret key. The reconstructed AES key is then used to encrypt each subset in CTR mode, and the resulting encrypted subsets are prepared for transmission to the master node (❺).

3.3. Parallel File Encryption in Distributed Architecture

Procedure: Figure 4 describes how our proposed scheme performs parallel file encryption across distributed slaves. After key transmission, the master calculates the start offset for each slave based on the total file size and the number of participating slaves. Each slave uses this offset information to perform file encryption independently.

As shown in the figure, each slave from 0 to N reads its assigned file segments from `offset_start` to `offset_end` in a streaming manner. The file is read in fixed-size subset units, and each subset is encrypted using AES-CTR within the `AES Encrypt`. The encrypted subsets are placed into a dedicated `Cipher Queue` on each slave. When the queue reaches the predefined batch size, encrypted subsets are grouped into a batch and asynchronously sent to the master using `MPI_Isend()`. This overlapping of encryption and communication enables continuous processing without idle resources.

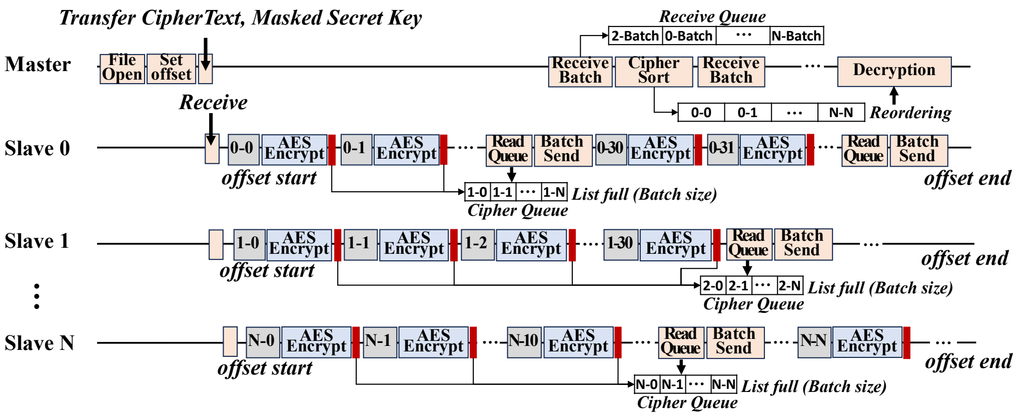


Figure 4. The procedure of offset-based parallel processing and transmission in distributed AES encryption.

After receiving encrypted subsets, the master appends them to the `Receive Queue`. Each slave sends multiple subsets in a batch, and the master collects them. `Receive Queue` manages incoming batches from all slaves. Subsets in the queue are then forwarded to the `Cipher Sorter`, which performs global reordering based on subset index sequence. For example, subsets (e.g., 1-0, 1-1 from slave 1 and N-0, N-1 from slave N) may arrive out of order, but `Cipher Sorter` restores the correct sequence before decryption. The sorted subsets are then processed in the `Decryption`. The AES key for decryption is reconstructed by first unmasking the PQC secret key using the QKD session key. The recovered secret key is then used to decapsulate the PQC ciphertext, yielding the AES key. This key is then used to decrypt all subsets in parallel using a thread pool.

Execution model: Algorithm 1 shows the overlapped execution model used by each slave node for AES encryption and subset transmission. Our proposed scheme separates file reading and subset sending into two concurrent paths, enabling encryption and communication to proceed in parallel. This design maximizes I/O throughput by minimizing idle time between subset production and transmission, and supports scalable performance even as the number of nodes or data size increases.

Algorithm 1 Overlapped AES encryption and subset transfer in slave node.

```
 1: Function SlaveProcess(filepath, offset_start, offset_end, aes_key)
 2: $current \leftarrow offset\_start$,    $subset\_idx \leftarrow offset\_start/subset\_size$
 3: Init CipherQueue ← dedicated queue,    StopToken ← object()
 4: Spawn Thread AESReaderThread()
 5: function AESReaderThread
 6:     Open file in binary mode
 7:     while $current < offset\_end$ do
 8:         $read\_size \leftarrow \min(subset\_size, offset\_end - current)$
 9:         $data \leftarrow$ read($read\_size$)
10:         $enc\_subset \leftarrow$ AES_Encrypt($data$, $aes\_key$)
11:         CipherQueue.put($subset\_idx$, $enc\_subset$)
12:         $current \leftarrow current + read\_size$,    $subset\_idx \leftarrow subset\_idx + 1$
13:     end while
14:     CipherQueue.put(StopToken)
15: end function

16: Init BatchBuffer ← empty list,    PendingRequests ← empty list
17: while True do
18:     $(i, c) \leftarrow$ CipherQueue.get()
19:     if $(i, c) ==$ StopToken then
20:         break
21:     end if
22:     BatchBuffer.append($i$, $c$)
23:     if len(BatchBuffer) == batch_size then
24:         $r \leftarrow$ MPI.isend(BatchBuffer, dest=0, tag=...)
25:         PendingRequests.append($r$),   BatchBuffer.clear()
26:         if len(PendingRequests) $\geq 32$ then
27:             MPI.Request.Waitall(PendingRequests),   PendingRequests.clear()
28:         end if
29:     end if
30: end while
31: if BatchBuffer not empty then
32:     MPI.isend(BatchBuffer, ...),   PendingRequests.append()
33: end if
34: MPI.Request.Waitall(PendingRequests)
```

As shown in the algorithm, the `SlaveProcess` function initializes the current file position and subset index **(Line 2)**. `CipherQueue` is created to buffer encrypted subsets between the reader and sender components **(Line 3)**, and a separate reader thread is spawned to handle encryption **(Line 4)**. The `AESReaderThread` function opens the target file in binary mode **(Line 6)** and sequentially reads the assigned region in fixed-size subset units **(Lines 7–8)**. Each subset is encrypted using the AES key reconstructed from the PQC ciphertext and QKD-masked secret key **(Line 10)**, and the resulting ciphertext is inserted into the `CipherQueue` along with its subset index **(Line 11)**. Once the entire file range `[offset_start, offset_end]` is processed, a `StopToken` is pushed into the queue to signal completion **(Line 14)**.

While the reader thread generates encrypted subsets, the main thread concurrently dequeues items from `CipherQueue` **(Line 17)** and appends them to a `BatchBuffer` **(Line 22)**. When the batch reaches a predefined size, a non-blocking send is triggered using `MPI_Isend()` **(Line 24)**, and the request handle is added to the `PendingRequests` list **(Line 25)**. When the number of outstanding requests exceeds a threshold (e.g., 32), `MPI_Waitall()` is invoked to ensure network buffer safety and avoid congestion **(Line 27)**. After all encrypted subsets are processed and the `StopToken` is detected **(Lines 19–20)**, any remaining batch is transmitted **(Lines 31–32)**, and a final `MPI_Waitall()` ensures completion of all asynchronous transmissions **(Line 34)**. Our proposed scheme leverages an overlapped pipeline design, allowing parallel encryption and transmission to proceed concurrently with minimal blocking. This enables efficient and scalable distributed encryption even under high I/O and communication loads.

3.4. Implementation

We implemented our proposed scheme by designing a parallel file encryption and centralized decryption framework that integrates post-quantum cryptography (PQC), quantum key distribution (QKD), and AES-based symmetric encryption across a distributed architecture for cloud systems. To support subset-level processing, overlapped encryption and transmission, and centralized decryption, we implemented approximately 270 lines of Python 3.9 code using `mpi4py`, `oqs-python`, `qiskit`, and `pycryptodome`.

Our implementation includes the following key components:

- **Layered Key Initialization:** We used the Kyber512 algorithm from the Open Quantum Safe (OQS) library [78] to generate a PQC key pair. The AES key is encapsulated using the PQC public key. The PQC secret key is XOR-masked using a QKD session key simulated with Qiskit Aer [62], and optionally split via Shamir's secret sharing for fault-tolerant recovery.
- **Parallel and Overlapped Subset Encryption:** Each slave node performs AES encryption and transmission in parallel by overlapping file reading, encryption, and communication. Encrypted subsets are streamed to the master using non-blocking MPI operations, allowing continuous processing without idle time. This structure improves resource utilization and enables high-throughput distributed encryption.
- **Centralized Decryption Pipeline:** The master node receives encrypted subsets, reorders them based on subset indices, and reconstructs the AES key by decapsulating the PQC ciphertext and unmasking the PQC secret key. Subsets are decrypted in parallel using a thread pool and merged to restore the original file.

We open-source our implementation at: https://github.com/changjongkim/Hybrid-Encrpytion-Scheme.git, accessed on 6 July 2025 .

4. Evaluation

4.1. Experimental Setup

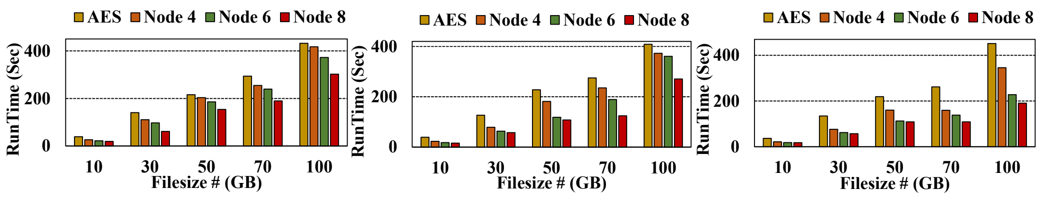
For evaluation, we used a cluster composed of 8 physical compute nodes, each equipped with an AMD EPYC 7713 processor (64 cores/128 threads @ 2.0 GHz), 130 GB of DDR4 memory, and a Seagate FireCuda 530 NVMe SSD with 2TB capacity. All nodes run Ubuntu 22.04.3 LTS with Linux kernel version 6.6.2. All nodes communicate through the Message Passing Interface (MPI).

To evaluate subset-level encryption under realistic conditions, we used the ECU-IoHT dataset [79]. We gradually increased the file size from 10 GB to 100 GB to assess scalability across various data volumes. For key management, we adopted the Kyber512 algorithm from the OQS library [78] for PQC-based encapsulation of a 256-bit AES key [76,77], and used the Qiskit Aer simulator [62] to emulate BB84-based QKD session key generation for secret key masking. To create a realistic evaluation scenario, we used a macro-benchmark from real application environments to assess the feasibility of our proposed scheme. We utilized the Yahoo Cloud Serving Benchmark (YCSB) [80], which is widely used for read/write-intensive workloads in cloud systems. For key-value storage, we configured

YCSB with RocksDB. During evaluation, we executed YCSB workloads A, B, C, and D under two separate conditions: once while running AES (Original) and once while running our proposed scheme. This allows for measuring and comparing the encryption overhead imposed on application-level performance.

4.2. Throughput

End-to-End: Figure 5 shows the end-to-end runtime performance of our proposed scheme compared to AES (Original) across different subset sizes (e.g., 4 MB, 8 MB, and 16 MB). The x-axis represents the input file size ranging from 10 GB to 100 GB, and the y-axis represents the total runtime in seconds. The legend distinguishes AES (baseline, single-node execution) and our proposed scheme evaluated on 4, 6, and 8 nodes (e.g., one master and multiple slaves), respectively. The end-to-end runtime for AES includes file read, encryption, decryption, and output generation on a single node. In contrast, our proposed scheme includes file read, quantum-resilient security, which includes PQC-based AES key encapsulation and QKD-based session key masking, parallel AES encryption, inter-node communication, decryption at the master, and the generation of the final output file.

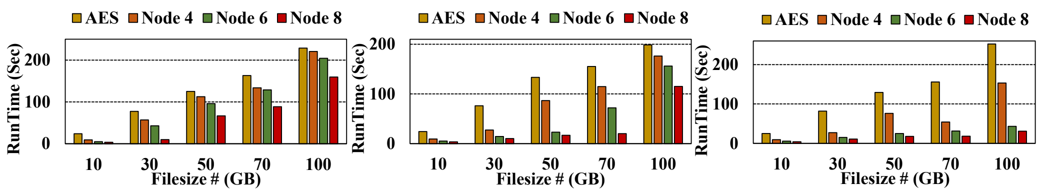


(**a**) 4 MB. (**b**) 8 MB. (**c**) 16 MB.

Figure 5. End-to-End Performance comparison between AES (Original) and Proposed (PQC + QKD + AES) across multi-node.

As shown in the figure, our proposed scheme consistently achieves lower runtime than the AES (Original) baseline across all subset sizes and file sizes. For 100 GB file, the runtime is reduced from 432.01 to 302.28 s with 4 MB, from 408.38 to 270.44 s with 8 MB, and from 451.86 to 190.86 s with 16 MB, achieving 1.43×, 1.51×, and 2.37× speedups, respectively. Overall, our proposed scheme shows 1.43× to 2.37× improvement over AES (Original), depending on the subset size. However, the end-to-end runtime does not scale linearly with the number of nodes, primarily because the decryption phase is centrally performed at the master node. The master node centrally manages the Kyber-decapsulated AES key and QKD-masked secret keys, with the goal of minimizing exposure to key compromise during recovery. Despite this centralized decryption overhead, our scheme shows consistent performance gains across all file sizes, demonstrating its effectiveness in balancing security requirements with parallel execution efficiency.

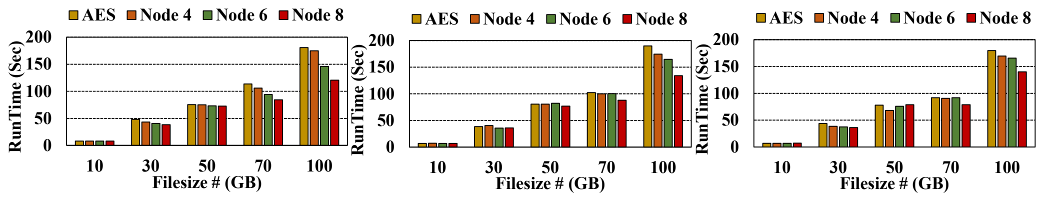
Encryption: Figure 6 shows the encryption-only runtime performance of our proposed scheme compared to AES (Original) across various subset sizes (e.g., 4, 8, and 16 MB). As shown in the figure, the encryption stage exhibits linear scalability as the number of nodes increases, enabled by offset-aware partitioning that assigns disjoint byte ranges to each slave, allowing AES encryption to proceed independently without inter-node synchronization and centralized bottlenecks. For 70 GB file, encryption is faster as more nodes are used. With 4 MB sizes, the encryption runtime is reduced from 163.34 s (AES) to 133.83, 128.83, and 88.61 s on 4, 6, and 8 nodes, respectively, achieving 1.26× to 1.84× speedups. With 8 MB, it is reduced from 155.16 (AES) to 114.67, 72.09, and 20.09 s, showing improvements of 1.35× to 7.72×. The most significant gain is observed with 16 MB, where the encryption runtime decreases from 155.92 (AES) to 54.25, 31.41, and 17.98 s, yielding 2.87× to 8.67× speedups depending on the number of nodes. A similar trend is observed for the 100 GB file. With 4 MB, runtime is reduced from 228.86 s (AES) to 220.67, 204.36, and 159.74 s on 4, 6, and 8 nodes, resulting in 1.04× to 1.43× speedups. With 8 MB, it is reduced from 198.73 (AES) to 176.16, 156.10, and 114.99 s, showing 1.13× to 1.73× improvements. For 16 MB, the runtime decreases from 252.62 (AES) to 153.41, 43.34, and 31.17 s, achieving 1.65× to 8.11× speedups.



(**a**) 4 MB. (**b**) 8 MB. (**c**) 16 MB.

Figure 6. Encryption Performance comparison between AES (Original) and Proposed (PQC + QKD + AES) across multi-node.

Decryption: Figure 7 shows the decryption runtime performance of our proposed scheme compared to AES (Original), evaluated with 100 GB input across different subset sizes (e.g., 4, 8, and 16 MB). As decryption is centrally performed at the master node, the decryption runtime is primarily bounded by the master's local execution, regardless of the number of slave nodes. With 4 MB, AES (Original) requires 180.71 s, while our scheme achieves 174.90, 146.11, and 120.58 s on 4, 6, and 8 nodes, respectively, yielding up to a 1.50× speedup. For 8 MB, the runtime is reduced from 189.93 AES (Original) to 174.50, 164.58, and 134.17 s, achieving up to a 1.41× improvement. With 16 MB, decryption time decreases from 179.52 AES (Original) to 169.65, 165.59, and 139.92 s, resulting in a maximum speedup of 1.28×. Although both the AES (Original) and our proposed scheme perform centralized, thread-based decryption, our scheme shows improved or comparable performance for large files. This is primarily due to structural overlap between communication, memory preparation, and computation. While the master node receives encrypted subsets over the network, it concurrently prepares in-memory buffers and sorts incoming data, allowing decryption to start immediately after reordering. This overlap reduces idle time and enables more efficient use of computational resources. In contrast, AES (Original) performs decryption directly on data loaded from disk, which introduces I/O latency that becomes more pronounced with larger file sizes, such as 70 and 100 GB. In addition, our proposed scheme adopts a streaming-based and offset-aware design to prevent memory exhaustion during decryption. Each decrypted subset is immediately written to the recovered file at the corresponding offset, without aggregating all data in memory. our design maintains stable and efficient memory usage regardless of dataset size and supports large-scale data processing.



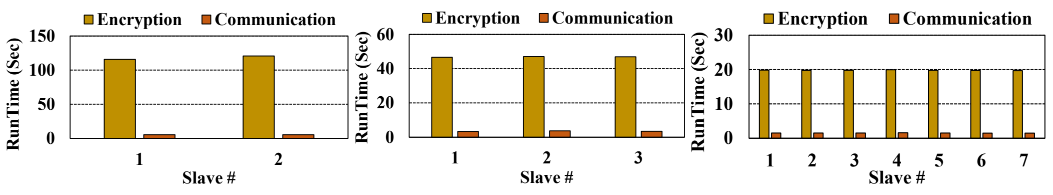
(**a**) 4 MB. (**b**) 8 MB. (**c**) 16 MB.

Figure 7. Decryption Performance comparison between AES (Original) and Proposed (PQC + QKD + AES) across multi-node.

4.3. Node-Level Encryption Performance and Variability

To evaluate the internal scalability, we examine how encryption and communication workloads are distributed across slave nodes as the number of participating nodes increases. As the number of nodes increases, additional slave nodes are introduced into the encryption pipeline. We analyze whether these slave nodes exhibit consistent encryption and communication performance, and whether increasing the number of nodes affects per-node performance variability or leads to workload imbalance.

Figure 8 shows the encryption and communication time for each slave under 2, 4, and 8-node configurations (excluding the master) using 70 GB Datasets. Each slave is responsible for encrypting its assigned subset independently using AES-CTR and transmitting the encrypted data back to the master via MPI. As shown in the figure, node-level encryption and communication times remain highly consistent across all configurations. For 3 Nodes (1 master and 2 slaves), the two slaves completed encryption in 115.89 and 120.79 s, and communication in 5.20 and 5.23 s, respectively. With 4 nodes, encryption times across slaves ranged from 46.66 to 47.06 s, and communication from 3.44 to 3.64 s. In the 8-node configuration, all slaves completed encryption within 19.65 to 19.95 s, and communication from 1.47 to 1.55 s.



(**a**) 3 Nodes. (**b**) 4 Nodes. (**c**) 8 Nodes.

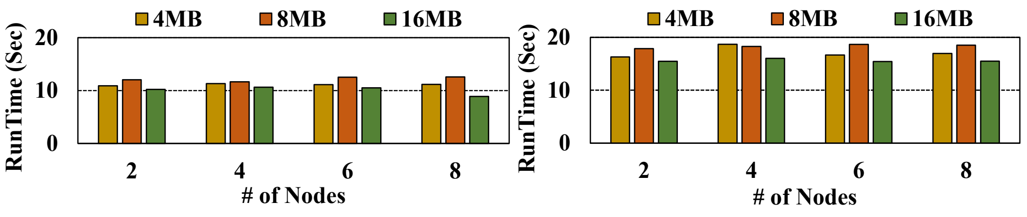
Figure 8. Node-Level encryption and communication time across different node configuration (1 master, all other nodes as slaves).

These results demonstrate that our offset-aware partitioning and parallel AES processing strategy achieve balanced workload distribution. The low variance in encryption time across slaves indicates that no single node becomes a bottleneck, even as the number of participating slaves increases. In addition, communication overhead increases predictably with node count, reflecting the benefits of using non-blocking MPI transfers and streaming batch transmission. Notably, increasing the number of nodes reduces per-node encryption time nearly linearly, as the total file is divided into smaller subsets. This ensures that the proposed design supports efficient scaling without incurring synchronization overhead or load imbalance.

4.4. Communication Cost

Figure 9 presents the communication time measured under varying numbers of nodes and subset sizes, using 70 GB and 100 GB files. The x-axis represents the number of nodes, and the y-axis shows the communication time in seconds required to transfer encrypted subsets from all slave nodes to the master via non-blocking MPI. As shown in the figure, the communication time remains consistently low across all configurations. For the 70 GB, the time ranges from 8.88 to 12.57 s, depending on the number of nodes and subset size. With 4 MB, the time ranges from 10.89 s (for 2 nodes) to 11.33 s (for 4 nodes), showing only a 1.04$\times$ difference. With 8 MB, the time range is from 11.66 s (for 4 nodes) to 12.57 s (for 8 nodes), corresponding to a 1.08$\times$ difference. For 16 MB, the time varies from 8.88 s (for 8 nodes) to 10.21 s (for 2 nodes), resulting in a 1.15$\times$ difference. For the 100 GB case, communication time ranges from 15.43 to 18.69 s. With 4 MB, the time ranges from 16.30 s (for 2 nodes) to 18.69 s (for 4 nodes), a 1.15$\times$ difference. With 8 MB, the cost ranges from 17.89 s (for 2 nodes) to 18.66 s (for 6 nodes), showing a 1.04$\times$ difference. For 16 MB, the time ranges from 15.43 s (for 2 nodes) to 16.06 s (for 4 nodes), also showing a 1.04$\times$ difference.

Across all subset sizes and file sizes, the variation in communication time remains minimal, demonstrating that communication overhead is both stable and scalable. This stability stems from our use of non-blocking MPI, which allows each slave node to transmit pre-encrypted subset data independently and in parallel. This is because all data is securely encrypted before transmission, eliminating the need for additional encryption or secure channel protocols during communication. Our design preserves security guarantees while keeping the communication layer lightweight and efficient.



(**a**) 70 GB. (**b**) 100 GB.

Figure 9. Communication time comparison of the proposed scheme (PQC + QKD + AES) with varying subset sizes across different node counts.

4.5. Time Analysis

It is critical for our proposed scheme to reduce processing overhead while maintaining secure communication and centralized coordination. To analyze the runtime composition of each subcomponent, Figure 10 presents the execution time breakdown when encrypting a 100GB file using 16 MB subset size across 2, 4, and 8 nodes. We observe the runtime of five major components: *Encryption*, *Decryption*, *BB84 Protocol*, *PQC Generation and Encapsulation*, and *Communication*.

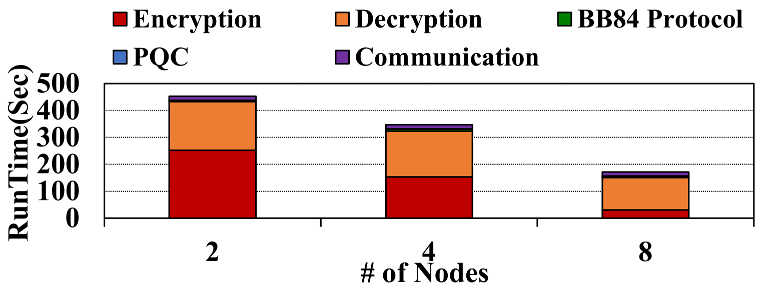


Figure 10. Time analysis using 100 GB file (subset: 16 MB).

As shown in the figure, *Encryption* accounts for the largest portion of runtime in the 2-node case, taking 252.62 s, which corresponds to 54.7% of the total time. As the number of nodes increases to 4 and 8, the encryption time decreases to 153.41 s (39.6%) and 31.17 s (11.8%), respectively. This corresponds to a 39.3% reduction from 2 to 4 nodes, and an 87.7% reduction from 2 to 8 nodes, demonstrating strong scalability through distributed subset encryption. In contrast, *Decryption* shows limited scalability due to its centralized execution at the master node. The decryption time decreases from 179.52 s at 2 nodes to 169.65 s (5.5% reduction) at 4 nodes and 119.92 s (33.2% reduction) at 8 nodes. Although decryption is centralized and not parallelized across nodes, our scheme mitigates potential bottlenecks by decoupling communication and computation. While encrypted subsets are being received, the master prepares in-memory buffers and schedules decryption tasks, enabling immediate processing after sorting is complete. This structure reduces idle time and improves overall throughput, especially for large files.

The *BB84 Protocol* time remains relatively small and does not show a clear correlation with the number of nodes. It varies between 4.23 and 7.23 s across configurations, primarily due to the probabilistic nature of quantum key generation rather than node-level scaling. This range indicates stable runtime behavior under simulated QKD conditions. The *PQC Encapsulation* time remains stable at approximately 1 s across all configurations. This is because key encapsulation is performed once during the initialization phase on the master node, independent of the number of slave nodes, and introduces negligible overhead within the overall encryption pipeline. Finally, *Communication* shows minimal variation, measured as 15.49 s at 2 nodes, 16.06 s at 4 nodes, and 15.51 s at 8 nodes. This stability comes from transmitting only pre-encrypted subset data using non-blocking MPI without additional protocol overhead. Since all data is encrypted before transmission and secured through PQC-based key encapsulation and QKD-based masking, no additional secure channel like TLS is required. As a result, communication cost remains low regardless of the number of nodes.

4.6. QKD Runtime Variability

Figure 11 shows the runtime variability of the QKD key generation phase across 25 independent BB84 protocol sessions. The x-axis represents the session number, the y-axis indicates the runtime in seconds, and the values inside each bar indicate the retry loop count to generate a valid key under the QBER threshold. Each session generates a 128-bit key using an identical quantum circuit structure and performs post-processing under the condition that the quantum bit error rate (QBER) remains below 11%. As shown in the figure, the runtime remains relatively stable across sessions. The minimum and maximum runtimes are 3.35 and 6.79 s, respectively, with an average of 4.49 s and a standard deviation of 0.88 s. Most sessions fall between 4 and 5 s, resulting in only a 2.03× difference between the fastest and slowest cases. QKD sessions with runtimes of 3 to 4, 4 to 5, and over 6 s completed in 4–5, 6–8, and up to 12 loops, respectively.

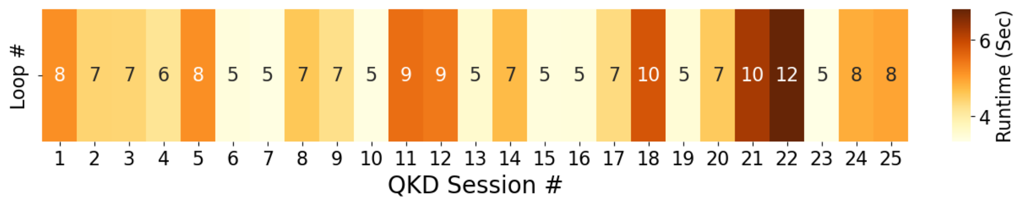


Figure 11. QKD Runtime Variability.

This consistent runtime shows that the QKD key generation latency is not significantly affected by the inherent randomness of the BB84 protocol. Within the full encryption routine of our proposed scheme, the QKD Key generation phase incurs negligible overhead and shows low variability, validating its practicality for repeated use in cloud systems. In addition, to ensure QBER compliance, the key generation loop is retried until the extracted key satisfies the <11% threshold. Despite this retry mechanism, the total runtime remains low due to parallel execution across multiple thread workers. This enables fast key agreement with minimal latency, making the QKD Key generation phase suitable for secure large-scale encryption without incurring significant performance overhead.

4.7. Computation Overhead

In many cloud systems, large-scale file-encryption tasks (e.g., securing logs, analytical datasets, or backups) are performed asynchronously in the background, while real-time computation workloads continue to run concurrently. Although small-sized file operations typically incur minimal and short-lived system costs (e.g., transient memory usage or localized I/O), large-scale file encryption can introduce sustained I/O pressure, memory contention, and CPU competition. To identify the practicality of our scheme in cloud systems, we evaluate whether background encryption interferes with concurrently running applications under realistic cloud workloads.

While our proposed scheme enhances security by integrating PQC and QKD with AES, it may affect overall performance by incurring computational overhead. To evaluate this impact, we measured the workload running time across four YCSB workloads: A, B, C, and D. Each workload reflects a distinct access pattern: workload A (update-heavy), B (read-mostly), C (read-only), and D (read-latest). In addition, each workload was executed using the YCSB with request distribution set to Zipfian. Three configurations were compared: a baseline YCSB workload execution with no encryption (Vanilla), execution under AES-only encryption (AES (Original)), and execution under our proposed scheme integrating PQC, QKD, and AES. In all configurations, the YCSB workload was executed on the identical master to ensure consistency. While Vanilla and AES (Original) performed encryption on a single node, our proposed scheme executed distributed encryption across 4 nodes, with the master handling the workload and coordination.

Table 2 presents the runtime results for all three configurations, averaged over five runs. As shown in the table, both AES (Original) and our proposed scheme increase workload processing time compared to Vanilla, but our proposed scheme consistently incurs lower overhead than AES (Original). In workload A, the runtime for Vanilla, AES, and Proposed is 255.65, 291.43, and 267.74 s, corresponding to a 14.01% increase for AES and 4.72% for Proposed. In workload B, the runtimes are 168.86, 209.65, and 182.34 s, resulting in a 24.17% increase for AES and 7.98% for Proposed. In workloads C and D, the runtime for Vanilla, AES, and Proposed is 115.55, 173.61, and 153.32 s in workload C, and 111.06, 146.62, and 132.17 s in workload D. These correspond to increases of 50.23% (AES) and 32.73% (Proposed) in workload C, and 31.98% (AES) and 19.02% (Proposed) in workload D, respectively. These results show that while both AES and our proposed scheme incur overhead compared to unencrypted execution, our proposed scheme consistently reduces the overhead impact, demonstrating its efficiency and practicality.

Table 2. Runtime of Vanilla, AES (Original), and Proposed (PQC + QKD + AES).

Workload	Vanilla	AES (Original)	Proposed (PQC + QKD + AES)
workload A	255.65	291.43	267.74
workload B	168.86	209.65	182.34
workload C	115.55	173.61	153.32
workload D	111.06	146.62	132.17

4.8. CPU Utilization

Figure 12 shows the average CPU utilization during encryption for AES (Original) and our proposed scheme with 4, 6, and 8 nodes. The x-axis represents runtime (seconds), and the y-axis shows CPU utilization in percentage (%). As shown in the figure, the AES (Original) exhibits higher variation and sustained CPU activity over a longer runtime. The execution completes in approximately 440 s, with CPU utilization ranging from 5.00% to 75.76%. This indicates uneven processing and frequent spikes due to the sequential workload on a single node. In contrast, our proposed scheme shows lower and more stable CPU usage with shorter runtimes across all configurations. Node 4 completes in 314 s with utilization ranging from 13.84% to 62.10%, Node 6 in 140 s with 26.73% to 59.54%, and Node 8 in 85 s with 20.80% to 63.46%. Compared to AES (Original), the CPU utilization patterns of our proposed scheme demonstrate improved load balancing and lower processing overhead across nodes.

These results demonstrate the advantages of our proposed scheme in terms of scalability and resource efficiency. Unlike AES (Original), which concentrates the entire encryption workload on a single node, our design distributes subset-encryption tasks across multiple nodes, enabling concurrent processing and reducing overall execution time. Despite utilizing more CPUs, our scheme maintains moderate and balanced usage across nodes, avoiding excessive load on any single core. This reduces bottlenecks and ensures that increasing the number of nodes leads to faster runtime without causing resource saturation. In addition, although CPU utilization remains moderate rather than saturated, such behavior does not indicate idle time or resource under-utilization. Rather, the reduced utilization reflects effective parallelism and reduced workload per node (e.g., in an 8-node processing a 100 GB file, each node encrypts only approximately 12.5 GB independently as a subset). The consistent runtime improvements across 4, 6, and 8 nodes confirm that performance scales with added resources without incurring communication bottlenecks or wait-time stalls.

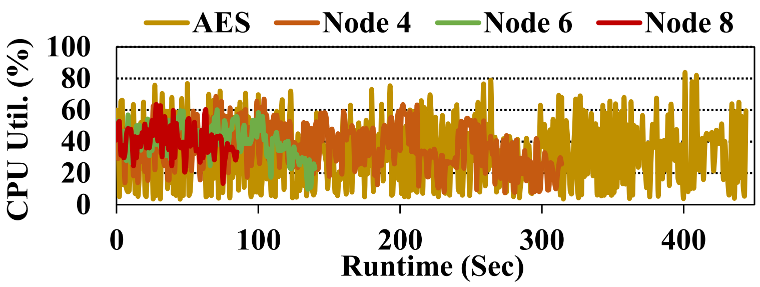


Figure 12. CPU utilization comparison between AES (Original) and our proposed scheme (PQC + QKD + AES) across multiple nodes.

5. Related Works

5.1. Cryptographic Integration Approaches for Cloud System Security

Various studies have been proposed for combining file encryption in cloud systems with advanced cryptographic algorithms to enhance security. Atikah et al. [81] proposed an encryption technique combining AES and RC4, leveraging the strong security of AES and the high processing speed of RC4 to improve the avalanche effect and overall file protection. An et al. [63] focused on increasing encryption performance for large-scale data through parallel optimization by combining XTS and AES, utilizing GPU-based parallelism to accelerate encryption and improve scalability. Abbas et al. [82] proposed combining AES and RSA encryption with steganography to hide encrypted data within image files, adding a layered security model that enhances confidentiality and integrity in cloud-based storage systems. Zeng et al. [37] proposed a hybrid PQC-QKD protocol for secure key distribution across networked systems. Their work constructs tree-based protocol compositions using XOR and secret sharing to evaluate key rate and vulnerability across multiple communication paths, focusing on protocol-level optimization in hybrid quantum-classical networks.

Our work is in line with these studies in proposing an encryption scheme for cloud systems. However, it also addresses the limitations of existing approaches by considering both classical and quantum security threats. By integrating Kyber-based post-quantum key encapsulation with BB84-based Quantum Key Distribution (QKD), our scheme provides a layered encryption architecture optimized for secure and scalable encryption in distributed cloud systems. While Zeng et al. [37] proposed a hybrid PQC-QKD protocol focusing on the key-distribution layer and protocol modeling, their work does not address practical encryption workflows or system-level integration for file protection. In contrast, our study designs a unified encryption framework for file-level protection that tightly integrates key generation, AES-based data encryption, and transmission coordination, enabling secure and scalable execution across cloud systems. Thus, our proposed scheme enables scalable and secure file protection by decoupling key management from data encryption, supporting parallel AES encryption across slave nodes while maintaining quantum-resilient key confidentiality through centralized coordination and multi-layered key masking.

5.2. Accelerating Security Processing in Cloud Systems with Innovative Cryptographic Methods

Several studies have designed various approaches to effectively accelerate the process of encrypting data in cloud systems. Velmurugadass et al. [83] proposed a blockchain-based security enhancement using elliptic curve encryption and cryptographic hash algorithms to protect IoT devices in cloud systems. Thabit et al. [84] proposed a lightweight encryption algorithm by combining block and stream encryption with optimal block and key sizes. Mohammed et al. [85] focused on reducing computational overhead by lowering operational costs in the data-encryption process through fast hashing and key-exchange mechanisms.

Our paper is in line with the goal of optimizing performance to accelerate encryption and decryption processes in cloud systems. However, our proposed scheme implements a layered encryption architecture that combines Kyber-based post-quantum key encapsulation and BB84-based quantum key distribution with parallel AES encryption. By assigning subset-level encryption tasks to slave nodes and overlapping encryption with transmission, it minimizes computational overhead and maximizes parallel efficiency. This structure enables consistent runtime performance regardless of file size, while the integration of a quantum-secure key masking layer mitigates vulnerabilities in traditional key exchange and enhances system scalability. In addition, although our scheme uses standard MPI channels without built-in security guarantees, all encryption keys are protected via Kyber encapsulation and QKD-based masking during distribution, and all subsets are transmitted in encrypted form. This eliminates the need for secure channels and ensures data confidentiality throughout the communication phase.

5.3. Integrated Cryptographic Approaches for Next-Generation Data Security

There have been studies exploring emerging encryption approaches to enhance privacy and security in image and sensitive data processing. Chen et al. [86] proposed an image-encryption method that combines the Piecewise Linear Chaotic Map (PWLCM) with the Standard Map to exploit both one-dimensional and two-dimensional chaotic behaviors for increased key sensitivity and diffusion. Mirzajani et al. [87] proposed a chaos-DNA hybrid model that utilizes DNA encoding and logic operations to further obscure the relationship between plaintext and ciphertext. Lin et al. [88] introduced a diversified memristive Hopfield Neural Network (HNN) that generates complex multi-butterfly chaotic attractors to secure Internet of Medical Things (IoMT) data. Similarly, Ding et al. [89] proposed a hidden multiwing HNN system that produces high-dimensional chaos suitable for encrypting remote sensing images. These studies exemplify a research trend focused on enhancing security by increasing algorithmic complexity within the classical computing domain.

Our paper is in line with these studies in the goal of designing advanced encryption schemes to provide robust security for specialized data, such as images and large-scale files. However, our study distinguishes itself by primarily addressing the security threats posed by quantum computing, a challenge not covered by the chaos-based classical approaches of these studies. By integrating Kyber-based Post-Quantum Cryptography (PQC) for key encapsulation and BB84-based Quantum Key Distribution (QKD) for key masking, our proposed scheme constructs a layered quantum-classical cryptographic framework designed to enhance security in distributed cloud systems. Additionally, our proposed scheme is designed for high-performance, scalable cloud systems. It enables secure and efficient large-scale file protection by decoupling key management from data encryption and supporting parallel AES encryption across distributed slave nodes. This is achieved through centralized coordination and multi-layered, quantum-resilient key masking, a systemic approach designed to balance next-generation security with practical throughput, unlike the algorithm-centric focus of the compared existing works.

6. Limitation and Future Works

Integrity verification mechanism: Our proposed scheme ensures safe key distribution and secure data delivery through a layered structure that combines Kyber-based key encapsulation, QKD-based masking, and optional Shamir secret sharing. (1) Each AES key is never exposed in plaintext. It is encapsulated via PQC, while the corresponding secret key is QKD-masked and optionally fragmented, making unauthorized key reconstruction infeasible even under partial node compromise. (2) All encrypted subsets are transmitted only after local encryption with securely reconstructed keys, and communication occurs over pre-encrypted data without relying on secure channels, thereby preventing eavesdropping and key leakage in practice. Additionally, each subset is encrypted in CTR mode using a randomly generated nonce, which supports freshness and prevents ciphertext duplication across sessions.

However, our study lacks cryptographic integrity verification to detect tampering or replay attacks on encrypted subsets or keys, such as when a compromised slave node encrypts and transmits manipulated data. Our plan for future work is to implement post-quantum digital signatures, such as Dilithium [61], to authenticate the origin and ensure tamper resistance of both key materials and encrypted subsets. To do this, each key and encrypted subset will be signed at the master and verified by each slave node, enabling full-path integrity checking from key generation to distributed encryption. In addition, we will bind the existing AES nonce to the digital signature or augment it with explicit timestamps to strengthen freshness guarantees and replay protection.

Non-elastic communication structure in cloud systems: Our proposed scheme adopts MPI-based communication between a master and fixed slave nodes (e.g., number of nodes: 2, 4, 6, 8), which has shown effective in stable, high-throughput performance under controlled cloud workloads such as YCSB (e.g., workload A, B, C, D). (1) This structure enables efficient coordination and predictable communication by leveraging deterministic message passing, resulting in minimal runtime variability across nodes and strong throughput scalability as demonstrated in our evaluation. (2) While this design meets the requirements of large-scale file encryption in a fixed-resource environment, it provides limited elasticity in dynamic cloud-native platforms such as Kubernetes [90], where autoscaling, container migration, and fault-tolerant recovery are essential.

Our plan for future work is to support elastic and fault-tolerant execution in cloud-native systems by modularizing communication and coordination using gRPC [91] and distributed key-value stores such as etcd. To achieve this, we will decouple encryption scheduling and metadata exchange from static MPI bindings, enabling seamless adaptation to elastic cloud infrastructure while maintaining the stable performance characteristics of the existing design.

Centralized decryption and performance-security trade-off: Our decryption strategy is performed at the master node using the AES key recovered through Kyber decapsulation and QKD unmasking, with both operations strictly confined to the master node to minimize key exposure and ensure coordinated trust. Although all slave nodes are capable of reconstructing the AES key from QKD-masked secret shares, only a selected slave nodes retain the key during execution, and the rest discard it immediately after encryption. As a result, decryption cannot be performed by slave nodes alone, which effectively eliminates unauthorized decryption paths and strengthens security against node compromise and replay-based attacks.

However, centralizing decryption introduces a performance bottleneck. As the number of slave nodes increases, the master node receives more encrypted subsets via MPI. For large files, this leads to concentrated communication, index-based reordering, and sequential decryption, creating I/O and compute contention that limits scalability. To address the limits of scalability, our planned future works will distribute decryption across multiple trusted nodes using threshold cryptography. Each node will handle only its assigned subsets using partial key shares, without access to the complete AES key. In addition, we plan to offload selective stages of decryption, such as AES pre-processing or partial decryption, to slave nodes under controlled policies that restrict key visibility, enforce task isolation, and prevent unauthorized reconstruction. This enables parallel execution while maintaining strict key confidentiality.

7. Conclusions

In this paper, we propose a secure and scalable file-encryption scheme that integrates quantum-resistant key encapsulation and quantum key distribution with distributed AES encryption for enhanced protection and parallel performance. Our evaluations show that the proposed scheme achieves up to 2.37$\times$ speedup in end-to-end runtime and up to 8.11$\times$ speedup in encryption time compared to AES (Original), while maintaining low communication cost, stable CPU utilization, and consistent QKD key generation latency. These results demonstrate its scalability and practicality for large-scale secure file processing in cloud systems, such as protecting medical archives, financial logs, or research datasets under background encryption.

To further extend our proposed scheme, we plan to integrate post-quantum signatures for subset integrity, enable elastic communication for dynamic cloud systems, and distribute decryption using threshold cryptography to improve scalability while preserving key confidentiality.

Author Contributions: Conceptualization, C.K.; Methodology, C.K. and S.K. (Seunghwan Kim); Software, C.K.; Validation, C.K.; Formal analysis, C.K. and S.K. (Seunghwan Kim); Investigation, C.K.; Resources, Y.S.; Data curation, C.K.; Writing—original draft, C.K.; Writing—review & editing, C.K. and S.K. (Sunggon Kim); Visualization, C.K.; Supervision, K.S., Y.S., M.K. and S.K. (Sunggon Kim); Project administration, S.K. (Sunggon Kim). All authors have read and agreed to the published version of the manuscript.

Funding: This research was supported by Seoul National University of Science & Technology.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The original contributions presented in this study are included in the article. Further inquiries can be directed to the corresponding author.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Kobusińska, A.; Leung, C.; Hsu, C.H.; S., R.; Chang, V. Emerging trends, issues and challenges in Internet of Things, Big Data and cloud computing. *Future Gener. Comput. Syst.* **2018**, *87*, 416–419. [CrossRef]
2. Ramirez, A.H.; Sulieman, L.; Schlueter, D.J.; Halvorson, A.; Qian, J.; Ratsimbazafy, F.; Loperena, R.; Mayo, K.; Basford, M.; Deflaux, N.; et al. The All of Us Research Program: Data quality, utility, and diversity. *Patterns* **2022**, *3*, 100570. [CrossRef] [PubMed]
3. Norori, N.; Hu, Q.; Aellen, F.M.; Faraci, F.D.; Tzovara, A. Addressing bias in big data and AI for health care: A call for open science. *Patterns* **2021**, *2*, 100347. [CrossRef]
4. Boubaker, S.; Liu, Z.; Zhai, L. Big data, news diversity and financial market crash. *Technol. Forecast. Soc. Change* **2021**, *168*, 120755. [CrossRef]
5. Hu, Y.; Kuang, W.; Qin, Z.; Li, K.; Zhang, J.; Gao, Y.; Li, W.; Li, K. Artificial intelligence security: Threats and countermeasures. *ACM Comput. Surv. (CSUR)* **2021**, *55*, 20. [CrossRef]
6. Abouelmehdi, K.; Beni-Hessane, A.; Khaloufi, H. Big healthcare data: Preserving security and privacy. *J. Big Data* **2018**, *5*, 1. [CrossRef]

7. Thomas, K.; Pullman, J.; Yeo, K.; Raghunathan, A.; Kelley, P.G.; Invernizzi, L.; Benko, B.; Pietraszek, T.; Patel, S.; Boneh, D.; et al. Protecting accounts from credential stuffing with password breach alerting. In Proceedings of the 28th USENIX Security Symposium (USENIX Security 19), Santa Clara, CA, USA, 14–16 August 2019; pp. 1556–1571.
8. Moallem, A. Human behavior in cybersecurity privacy and trust. In *Human-Computer Interaction in Intelligent Environments*; CRC Press: Boca Raton, FL, USA, 2024; pp. 77–107.
9. Wu, H.; Dwivedi, A.D.; Srivastava, G. Security and privacy of patient information in medical systems based on blockchain technology. *ACM Trans. Multimed. Comput. Commun. Appl. (TOMM)* **2021**, *17*, 60. [CrossRef]
10. Marjani, M.; Nasaruddin, F.; Gani, A.; Karim, A.; Hashem, I.A.T.; Siddiqa, A.; Yaqoob, I. Big IoT data analytics: Architecture, opportunities, and open research challenges. *IEEE Access* **2017**, *5*, 5247–5261.
11. Alansari, Z.; Soomro, S.; Belgaum, M.R.; Shamshirband, S. The rise of Internet of Things (IoT) in big healthcare data: Review and open research issues. In *Progress in Advanced Computing and Intelligent Engineering, Proceedings of ICACIE 2016, Puducherry, India, 15–17 December 2016*; Springer: Singapore, 2018; Volume 2, pp. 675–685.
12. Amazon Web Services (AWS). Available online: https://aws.amazon.com (accessed on 6 July 2025).
13. Google Cloud. Available online: https://cloud.google.com/ (accessed on 6 July 2025).
14. Microsoft Azure. Available online: https://azure.microsoft.com/ (accessed on 6 July 2025).
15. Dierks, T.; Rescorla, E. *The Transport Layer Security (TLS) Protocol Version 1.2*; Technical Report RFC 5246; Internet Engineering Task Force: Fremont, CA, USA, 2008.
16. Indu, I.; Anand, P.R.; Bhaskar, V. Identity and access management in cloud environment: Mechanisms and challenges. *Eng. Sci. Technol. Int. J.* **2018**, *21*, 574–588. [CrossRef]
17. Tom, J.J.; Anebo, N.P.; Onyekwelu, B.A.; Wilfred, A.; Eyo, R. Quantum computers and algorithms: A threat to classical cryptographic systems. *Int. J. Eng. Adv. Technol* **2023**, *12*, 25–38. [CrossRef]
18. Azhari, R.; Salsabila, A.N. Analyzing the impact of quantum computing on current encryption techniques. *IAIC Trans. Sustain. Digit. Innov. (ITSDI)* **2024**, *5*, 148–157. [CrossRef]
19. Ajala, O.A.; Arinze, C.A.; Ofodile, O.C.; Okoye, C.C.; Daraojimba, A.I. Exploring and reviewing the potential of quantum computing in enhancing cybersecurity encryption methods. *Magna Sci. Adv. Res. Rev* **2024**, *10*, 321–329. [CrossRef]
20. Shor, P.W. Algorithms for quantum computation: Discrete logarithms and factoring. In Proceedings of the 35th Annual Symposium on Foundations of Computer Science, Santa Fe, NM, USA, 20–22 November 1994; pp. 124–134.
21. Grover, L.K. A fast quantum mechanical algorithm for database search. In Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing, Philadelphia, PA, USA, 22–24 May 1996; pp. 212–219.
22. Daemen, J.; Rijmen, V. AES Proposal: Rijndael. NIST AES Proposal Document. 1999. Available online: https://csrc.nist.gov/csrc/media/projects/cryptographic-standards-and-guidelines/documents/aes-development/rijndael-ammended.pdf (accessed on date 6 July 2025).
23. Bernstein, D.J.; Lange, T. Post-quantum cryptography. *Nature* **2017**, *549*, 188–194. [CrossRef]
24. Scarani, V.; Bechmann-Pasquinucci, H.; Cerf, N.J.; Dušek, M.; Lütkenhaus, N.; Peev, M. The security of practical quantum key distribution. *Rev. Mod. Phys.* **2009**, *81*, 1301–1350. [CrossRef]
25. Jenefa, A.; Josh, F.; Taurshia, A.; Kumar, K.R.; Kowsega, S.; Naveen, E. PQC Secure: Strategies for defending against quantum threats. In Proceedings of the 2023 2nd International Conference on Automation, Computing and Renewable Systems (ICACRS), Pudukkottai, India, 13–15 December 2023; pp. 1799–1804.
26. Joseph, D.; Misoczki, R.; Manzano, M.; Tricot, J.; Pinuaga, F.D.; Lacombe, O.; Leichenauer, S.; Hidary, J.; Venables, P.; Hansen, R. Transitioning organizations to post-quantum cryptography. *Nature* **2022**, *605*, 237–243. [CrossRef]
27. Käppler, S.A.; Schneider, B. Post-quantum cryptography: An introductory overview and implementation challenges of quantum-resistant algorithms. *Proc. Soc.* **2022**, *84*, 61–71.
28. Sharma, P.; Agrawal, A.; Bhatia, V.; Prakash, S.; Mishra, A.K. Quantum key distribution secured optical networks: A survey. *IEEE Open J. Commun. Soc.* **2021**, *2*, 2049–2083. [CrossRef]
29. Zhang, W.; van Leent, T.; Redeker, K.; Garthoff, R.; Schwonnek, R.; Fertig, F.; Eppelt, S.; Rosenfeld, W.; Scarani, V.; Lim, C.C.W.; et al. A device-independent quantum key distribution system for distant users. *Nature* **2022**, *607*, 687–691. [CrossRef]
30. Tsai, C.W.; Yang, C.W.; Lin, J.; Chang, Y.C.; Chang, R.S. Quantum key distribution networks: Challenges and future research issues in security. *Appl. Sci.* **2021**, *11*, 3767. [CrossRef]
31. Sun, S.; Huang, A. A review of security evaluation of practical quantum key distribution system. *Entropy* **2022**, *24*, 260. [CrossRef]
32. Farooq, S.; Altaf, A.; Iqbal, F.; Thompson, E.B.; Vargas, D.L.R.; Díez, I.d.l.T.; Ashraf, I. Resilience optimization of post-quantum cryptography key encapsulation algorithms. *Sensors* **2023**, *23*, 5379. [CrossRef] [PubMed]
33. Hecht, P. PQC: R-Propping of Burmester-Desmedt Conference Key Distribution System. Cryptology ePrint Archive 2021. Available online: https://eprint.iacr.org/2021/024 (accessed on 6 July 2025).
34. Campbell, R. The need for cyber resilient enterprise distributed ledger Risk Management Framework. *J. Br. Blockchain Assoc.* **2020**, *3*, 9. [CrossRef] [PubMed]

35. Bouda, J.; Pivoluska, M.; Plesch, M.; Wilmott, C. Weak randomness seriously limits the security of quantum key distribution. *Phys. Rev. A Atomic Mol. Opt. Phys.* **2012**, *86*, 062308. [CrossRef]
36. Jiang, X.L.; Deng, X.Q.; Wang, Y.; Lu, Y.F.; Li, J.J.; Zhou, C.; Bao, W.S. Weak randomness analysis of measurement-device-independent quantum key distribution with finite resources. *Photonics* **2022**, *9*, 356. [CrossRef]
37. Zeng, P.; Bandyopadhyay, D.; Méndez, J.A.M.; Bitner, N.; Kolar, A.; Solomon, M.T.; Ye, Z.; Rozpędek, F.; Zhong, T.; Heremans, F.J.; et al. Practical hybrid PQC-QKD protocols with enhanced security and performance. *arXiv* **2024**, arXiv:2411.01086.
38. Yang, Z.; Shi, Q.; Cheng, T.; Wang, X.; Zhang, R.; Yu, L. A security-enhanced authentication scheme for quantum-key-distribution (QKD) enabled Internet of vehicles in multi-cloud environment. *Veh. Commun.* **2024**, *48*, 100789. [CrossRef]
39. Zeydan, E.; Baranda, J.; Mangues-Bafalluy, J. Post-quantum blockchain-based secure service orchestration in multi-cloud networks. *IEEE Access* **2022**, *10*, 129520–129530. [CrossRef]
40. Ricci, S.; Dobias, P.; Malina, L.; Hajny, J.; Jedlicka, P. Hybrid keys in practice: Combining classical, quantum and post-quantum cryptography. *IEEE Access* **2024**, *12*, 23206–23219. [CrossRef]
41. Wang, L.J.; Zhang, K.Y.; Wang, J.Y.; Cheng, J.; Yang, Y.H.; Tang, S.B.; Yan, D.; Tang, Y.L.; Liu, Z.; Yu, Y.; et al. Experimental authentication of quantum key distribution with post-quantum cryptography. *npj Quantum Inf.* **2021**, *7*, 67. [CrossRef]
42. Rani, A.; Ai, X.; Gupta, A.; Adhikari, R.S.; Malaney, R. Combined Quantum and Post-Quantum Security for Earth-Satellite Channels. In Proceedings of the 2025 International Conference on Quantum Communications, Networking, and Computing (QCNC), Nara, Japan, 31 March–2 April 2025; pp. 301–308.
43. Rivest, R.L.; Shamir, A.; Adleman, L. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM* **1978**, *21*, 120–126. [CrossRef]
44. Koblitz, N. Elliptic curve cryptosystems. *Math. Comput.* **1987**, *48*, 203–209. [CrossRef]
45. Faruk, M.J.H.; Tahora, S.; Tasnim, M.; Shahriar, H.; Sakib, N. A review of quantum cybersecurity: Threats, risks and opportunities. In Proceedings of the 2022 1st International Conference on AI in Cybersecurity (ICAIC), Virtual, 21–23 September 2022; pp. 1–8.
46. Szatmáry, S. Quantum Computers—Security Threats and Solutions. In Proceedings of the IFIP International Conference on Human Choice and Computers, Tokyo, Japan, 8–9 September 2022; pp. 431–441.
47. Kilber, N.; Kaestle, D.; Wagner, S. Cybersecurity for quantum computing. *arXiv* **2021**, arXiv:2110.14701.
48. Chawla, D.; Mehra, P.S. A survey on quantum computing for internet of things security. *Procedia Comput. Sci.* **2023**, *218*, 2191–2200. [CrossRef]
49. Kumar, M.; Pattnaik, P. Post quantum cryptography (pqc)-an overview. In Proceedings of the 2020 IEEE High Performance Extreme Computing Conference (HPEC), Virtual, 22–24 September 2020; pp. 1–9.
50. Soni, D.; Karri, R. Efficient hardware implementation of pqc primitives and pqc algorithms using high-level synthesis. In Proceedings of the 2021 IEEE Computer Society Annual Symposium on VLSI (ISVLSI), Tampa, FL, USA, 7–9 July 2021; pp. 296–301.
51. Micciancio, D.; Regev, O. Lattice-based cryptography. In *Post-Quantum Cryptography*; Springer: Berlin/Heidelberg, Germany, 2009; pp. 147–191.
52. Peikert, C. Public-key cryptosystems from the worst-case shortest vector problem. In Proceedings of the Forty-First Annual ACM Symposium on Theory of Computing, Bethesda, MD, USA, 31 May–2 June 2009; pp. 333–342.
53. Bos, J.; Ducas, L.; Kiltz, E.; Lepoint, T.; Lyubashevsky, V.; Schanck, J.M.; Schwabe, P.; Seiler, G.; Stehlé, D. CRYSTALS-Kyber: A CCA-secure module-lattice-based KEM. In Proceedings of the 2018 IEEE European Symposium on Security and Privacy (EuroS&P), London, UK, 24–26 April 2018; pp. 353–367.
54. Shor, P.W.; Preskill, J. Simple proof of security of the BB84 quantum key distribution protocol. *Phys. Rev. Lett.* **2000**, *85*, 441. [CrossRef]
55. Boneh, D. The decision diffie-hellman problem. In Proceedings of the International Algorithmic Number Theory Symposium, Leiden, The Netherlands, 2–7 July 2000; pp. 48–63.
56. Gerhardt, I.; Liu, Q.; Lamas-Linares, A.; Skaar, J.; Kurtsiefer, C.; Makarov, V. Full-field implementation of a perfect eavesdropper on a quantum cryptography system. *Nat. Commun.* **2011**, *2*, 349. [CrossRef]
57. Sibson, P.; Erven, C.; Godfrey, M.; Miki, S.; Yamashita, T.; Fujiwara, M.; Sasaki, M.; Terai, H.; Tanner, M.G.; Natarajan, C.M.; et al. Chip-based quantum key distribution. *Nat. Commun.* **2017**, *8*, 13984. [CrossRef] [PubMed]
58. Yang, S.S.; Bai, Z.L.; Wang, X.Y.; Li, Y.M. FPGA-based implementation of size-adaptive privacy amplification in quantum key distribution. *IEEE Photonics J.* **2017**, *9*, 7600308. [CrossRef]
59. Amiri, R.; Wallden, P.; Kent, A.; Andersson, E. Secure quantum signatures using insecure quantum channels. *Phys. Rev. A* **2016**, *93*, 032325. [CrossRef]
60. Amer, O.; Krawec, W.O. Semiquantum key distribution with high quantum noise tolerance. *Phys. Rev. A* **2019**, *100*, 022319. [CrossRef]
61. Ducas, L.; Kiltz, E.; Lepoint, T.; Lyubashevsky, V.; Schwabe, P.; Seiler, G.; Stehlé, D. Crystals-dilithium: A lattice-based digital signature scheme. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* **2018**, *2018*, 238–268. [CrossRef]
62. IBM Qiskit Aer. Available online: https://github.com/Qiskit/qiskit-aer (accessed on 6 July 2025).

63. An, S.; Seo, S.C. Designing a new XTS-AES parallel optimization implementation technique for fast file encryption. *IEEE Access* **2022**, *10*, 25349–25357. [CrossRef]
64. Baladhay, J.S.; Gamido, H.V.; Edjie, M. Large file encryption in a Reduced-Round Permutation-Based AES file management system. *Indones. J. Electr. Eng. Comput. Sci.* **2024**, *34*, 2021–2031. [CrossRef]
65. Xing, B.; Wang, D.; Yang, Y.; Wei, Z.; Wu, J.; He, C. Accelerating DES and AES algorithms for a heterogeneous many-core processor. *Int. J. Parallel Program.* **2021**, *49*, 463–486. [CrossRef]
66. Lipmaa, H.; Rogaway, P.; Wagner, D. CTR-mode encryption. In Proceedings of the First NIST Workshop on Modes of Operation, Gaithersburg, MD, USA, 20–21 October 2000; Volume 39.
67. Vaidehi, M.; Rabi, B.J. Design and analysis of AES-CBC mode for high security applications. In Proceedings of the Second International Conference on Current Trends In Engineering and Technology-ICCTET 2014, Coimbatore, India, 8–10 July 2014; pp. 499–502.
68. Lin, C.H.; Hu, G.H.; Chan, C.Y.; Yan, J.J. Chaos-based synchronized dynamic keys and their application to image encryption with an improved AES algorithm. *Appl. Sci.* **2021**, *11*, 1329. [CrossRef]
69. Ashraf, Z.; Sohail, A.; Yousaf, M. Robust and lightweight symmetric key exchange algorithm for next-generation IoE. *Internet Things* **2023**, *22*, 100703. [CrossRef]
70. Bogdanov, A.; Mendel, F.; Regazzoni, F.; Rijmen, V.; Tischhauser, E. ALE: AES-based lightweight authenticated encryption. In Proceedings of the Fast Software Encryption: 20th International Workshop, FSE 2013, Singapore, 11–13 March 2013; pp. 447–466.
71. Fei, X.; Li, K.; Yang, W.; Li, K. Practical parallel AES algorithms on cloud for massive users and their performance evaluation. *Concurr. Comput. Pract. Exp.* **2016**, *28*, 4246–4263. [CrossRef]
72. Awan, I.A.; Shiraz, M.; Hashmi, M.U.; Shaheen, Q.; Akhtar, R.; Ditta, A. Secure framework enhancing AES algorithm in cloud computing. *Secur. Commun. Netw.* **2020**, *2020*, 8863345. [CrossRef]
73. Liu, B.; Baas, B.M. Parallel AES encryption engines for many-core processor arrays. *IEEE Trans. Comput.* **2011**, *62*, 536–547. [CrossRef]
74. Shamir, A. How to share a secret. *Commun. ACM* **1979**, *22*, 612–613. [CrossRef]
75. Pang, L.J.; Wang, Y.M. A new (t, n) multi-secret sharing scheme based on Shamir's secret sharing. *Appl. Math. Comput.* **2005**, *167*, 840–848. [CrossRef]
76. Sarah, D.; Peter, C. On the practical cost of Grover for AES key recovery. In Proceedings of the Presentation at the 5th NIST PQC Standardization Conference, Rockville, MD, USA, 10–12 April 2024.
77. Grassl, M.; Langenberg, B.; Roetteler, M.; Steinwandt, R. Applying Grover's algorithm to AES: Quantum resource estimates. In Proceedings of the International Workshop on Post-Quantum Cryptography, Fukuoka, Japan, 24–26 February 2016; pp. 29–43.
78. Open Quantum Safe. Available online: https://openquantumsafe.org/ (accessed on 6 July 2025).
79. Ahmed, M.; Byreddy, S.; Nutakki, A.; Sikos, L.F.; Haskell-Dowland, P. ECU-IoHT: A dataset for analyzing cyberattacks in Internet of Health Things. *Ad Hoc Netw.* **2021**, *122*, 102621. [CrossRef]
80. Cooper, B.F.; Silberstein, A.; Tam, E.; Ramakrishnan, R.; Sears, R. Benchmarking cloud serving systems with YCSB. In Proceedings of the 1st ACM Symposium on Cloud Computing, Indianapolis, IN, USA, 10–11 June 2010; pp. 143–154.
81. Atikah, N.; Ashila, M.R.; Rachmawanto, E.H.; Sari, C.A. AES-RC4 Encryption Technique to Improve File Security. In Proceedings of the 2019 Fourth International Conference on Informatics and Computing (ICIC), Semarang, Indonesia, 16–17 October 2019; pp. 1–5.
82. Abbas, M.S.; Mahdi, S.S.; Hussien, S.A. Security improvement of cloud data using hybrid cryptography and steganography. In Proceedings of the 2020 international conference on computer science and software engineering (CSASE), Duhok, Iraq, 16–18 February 2020; pp. 123–127.
83. Velmurugadass, P.; Dhanasekaran, S.; Anand, S.S.; Vasudevan, V. Enhancing Blockchain security in cloud computing with IoT environment using ECIES and cryptography hash algorithm. *Mater. Today Proc.* **2021**, *37*, 2653–2659. [CrossRef]
84. Thabit, F.; Alhomdy, S.; Al-Ahdal, A.H.; Jagtap, S. A new lightweight cryptographic algorithm for enhancing data security in cloud computing. *Glob. Transitions Proc.* **2021**, *2*, 91–99. [CrossRef]
85. Mohammed, S.; Nanthini, S.; Krishna, N.B.; Srinivas, I.V.; Rajagopal, M.; Kumar, M.A. A new lightweight data security system for data security in the cloud computing. *Meas. Sens.* **2023**, *29*, 100856. [CrossRef]
86. Chen, Y.; Tang, C.; Yi, Z. A novel image encryption scheme based on PWLCM and standard map. *Complexity* **2020**, *2020*, 3026972. [CrossRef]
87. Mirzajani, S.; Moafimadani, S.S.; Roohi, M. A New Encryption Algorithm Utilizing DNA Subsequence Operations for Color Images. *AppliedMath* **2024**, *4*, 1382–1403. [CrossRef]
88. Lin, H.; Deng, X.; Yu, F.; Sun, Y. Diversified butterfly attractors of memristive HNN with two memristive systems and application in IoMT for privacy protection. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* **2024**, *44*, 304–316. [CrossRef]
89. Ding, S.; Lin, H.; Deng, X.; Yao, W.; Jin, J. A hidden multiwing memristive neural network and its application in remote sensing data security. *Expert Syst. Appl.* **2025**, *277*, 127168. [CrossRef]

90. Kubernetes. Available online: https://kubernetes.io (accessed on 6 July 2025).
91. Google gRPC. Available online: https://cloud.google.com/api-gateway/docs/grpc-overview (accessed on 7 July 2025).