Cluster Computing (2025) 28:464
https://doi.org/10.1007/s10586-025-05433-7

Cloudlock: secure data sharing using a hybrid cryptosystem in multi-cloud data storage

Nitesh Bharot[1] · Nakul Mehta[1] · John G. Breslin[1] · Priyanka Verma[2]

Received: 21 February 2025 / Revised: 1 May 2025 / Accepted: 2 May 2025 / Published online: 31 July 2025

Abstract
The advent of cloud computing has revolutionized data management, providing scalability and cost-effectiveness. However, it also presents challenges in ensuring data security and privacy. Thus organizations increasingly adopt multi-cloud storage strategies to avoid data loss and single points of failure, but a new array of complexities emerges, particularly related to secure data access and management with a multi-cloud strategy. This paper explores these complexities and presents an innovative encryption-based solution for secure data storage and easy accessibility in a multi-cloud system. In this work, we proposed CloudLock framework which uses hybrid ChaCha20-Poly1305 encryption mechanism to store and access the data in multi-cloud systems. Moreover, the secret key used for encryption and decryption is shared between the data owner and authentic users using Elliptic Curve Diffie-Hellman (ECDH) mechanism. Our focus is to establish a robust mechanism that permits only authenticated users to access and manage their data across multi-cloud platforms. Thus with the proposed model, we aim to balance security, accessibility, and efficiency, thereby offering a potential solution to multi-cloud storage's multifaceted challenges. Our rigorous security assessment and performance analysis, which considers a broad range of parameters including encryption and decryption time, upload and download time, turn-around time, and memory usage, affirms that our proposed framework is highly efficient. The framework meets the stringent requirements necessary for secure data sharing, demonstrating robust resilience to various security threats. When compared to existing methods, our proposed framework showcases superior performance and enhanced security characteristics.

Keywords Multi-cloud · Encryption · Secure data sharing · ChaCha20-Poly1305 · Cloud storage services

1 Introduction

The global cloud computing market size was estimated at USD 602.31 billion in 2023 and is expected to grow at a CAGR of 21.2% from 2024 to 2030 [1]. Cloud storage platforms offer scalability, flexibility, and cost-effectiveness, which are instrumental for businesses and organizations to handle the increasing data volume. However, alongside these benefits, there are also challenges in terms of data security, integrity, and data privacy [2–5]. Moreover, maintenance of single cloud platforms requires high costs which are not sustainable by most organizations. To address these challenges, organizations are increasingly adopting multi-cloud strategies. The global multi-cloud management market size was valued at USD 8.03 billion in 2022 and is expected to expand at a compound annual growth rate (CAGR) of 28.0% from 2023 to 2030 [6]. Multi-cloud services [7] leverage multiple cloud providers, reducing dependency on a single vendor and enhancing security by distributing data across various environments. This approach mitigates the risk of data loss, prevents a single point of failure, and strengthens compliance with diverse regulatory frameworks [8, 9].

✉ Priyanka Verma
priyanka.verma@universityofgalway.ie

Nitesh Bharot
nitesh.bharot@universityofgalway.ie

Nakul Mehta
n.mehta3@universityofgalway.ie

John G. Breslin
john.breslin@universityofgalway.ie

1 Data Science Institute, University of Galway, Galway H91TK33, Ireland

2 School of Computer Science, University of Galway, University Road, Galway H91TK33, Ireland

A multi-cloud storage environment has termed the use of two or more cloud computing platforms from different cloud vendors. Multi-cloud storage environments are designed to eliminate the reliance on a single cloud service provider, offering organizations a greater degree of flexibility and control over their data. One of the key benefits of a multi-cloud storage environment is risk mitigation. By spreading data across multiple platforms, organizations can prevent a single point of failure. This means if one cloud provider experiences an outage or a security breach, the organization can still store and operate its data from the other cloud platforms. Multi-cloud storage environments also offer performance optimization. Different cloud providers have various strengths and specialties, and organizations can take advantage of this diversity to optimize their operations. For instance, they might choose one cloud provider for AI-related tasks due to its superior machine learning capabilities and another for its efficient data storage solutions. Cost efficiency is another advantage of a multi-cloud storage environment. Different providers have different pricing structures, and organizations can strategically distribute their workloads to take advantage of the most cost-effective solutions.

The security of data housed in multi-cloud architectures is of paramount importance and is typically ensured using various methods, as detailed by Bohli et al. [10]. Secure data sharing across different cloud platforms presents several challenges, including data privacy, interoperability, and scalability. Sensitive data distributed across multiple clouds is vulnerable to unauthorized access, data breaches, and insider threats, making robust security mechanisms essential. Notably, the study by Khasim Shaik et al. [11] provides an analytical review of the most crucial security encryption algorithms designed to safeguard data within cloud computing. The inherent design of multi-cloud architectures presents an intrinsic obstacle to potential attackers, bolstering the security of stored information [12].

This work provides insights into the security paradigms of multi-cloud storage and contributes to existing research. It paves the way for the development of more secure, reliable, and user-friendly multi-cloud storage systems, fostering a safer and more efficient digital environment for organizations across the globe.

Several solutions have been proposed to facilitate the secure sharing of unstructured data in a multi-cloud scenario, as illustrated by studies conducted by many researchers [13–18]. However, these studies have not sufficiently addressed the need for a dependable and reliable architecture. Existing strategies in multi-cloud storage fail to guarantee protection against issues related to secure key distribution, key management [19], threats from malicious insiders and files, and the efficiency of the encryption algorithm used for secure data storing and accessing in the multi-cloud environment.

Furthermore, conflicts arising from file merging during the retrieval process can undermine data integrity, especially since indexed-based cryptographic data splitting is not employed in current approaches. The use of AES128-bit encryption, a common practice in many methodologies, also presents challenges. Performance response time is consequently impacted as the file size grows and the encryption procedure and the memory consumption takes longer to complete.

In spite of these challenges, as we continue to embrace digital transformation, multi-cloud storage environments are becoming increasingly significant. They offer a more resilient, flexible, and potentially cost-effective solution for data management in the era of big data and cloud computing.

Through this approach, we attempt to probe into these complexities, proposing an advanced solution to secure data storage and access in multi-cloud environments. Thus in this work, we proposed a CloudLock framework which uses hybrid ChaCha20-Poly1305 encryption mechanism for secure data sharing among authenticated users and secure data storage in multi-cloud environment. However, integrating this methodology into a multi-cloud storage system, while ensuring smooth access to authenticated users in a secure way through the use of Elliptic Curve Diffie-hellman (ECDH) within a Key Distribution System (KDS) constitutes a relatively uncharted field of research. The main contributions of this work are:

- Proposed CloudLock framework, a secure data sharing for a multi-cloud storage framework to provide a robust and reliable structure that can act as a middle ware between the data owner, authenticated users, and multi-cloud environment.
- An encryption-decryption mechanism for data storage leveraging the ChaCha20-Poly1305 algorithm. This approach takes advantage of the algorithm's efficient and high-speed cryptographic capabilities, enabling secure data management while reducing computational overhead.
- A key-sharing scheme between the data owner and authenticated users is facilitated through the application of an Elliptic Curve Diffie-Hellman (ECDH) mechanism for sharing the secret key used for encryption in the proposed CloudLock framework.

The rest of the paper is organized as: Section 2 discusses the related work. Section 3 presents the system model, assumptions, and security goals. Section 4 describes the preliminaries followed by Sect. 5 describing the proposed framework. Section 6 scrutinizes the results and analysis followed by Sect. 7 which concludes the work ad gives future direction.

2 Related work

This section presents the state of art work done in the related field and Table 1 shows the comparison and summary of these works. The domains of privacy and security in cloud storage have been extensively researched given their broad implications. There exist a multitude of vulnerabilities associated with file sharing over cloud platforms, which can be exploited for unauthorized access. Various malevolent intents behind such attacks can tarnish the reputation of cloud service providers. Karnik et al. [20] proposes a secure multi-cloud storage technique using data fragmentation, encryption, and hash-based integrity checks. It enhances confidentiality, availability, and fault tolerance while reducing latency and storage overhead, making it suitable for sensitive data in distributed cloud environments.

To augment secure data sharing within multi-cloud storage, an approach featuring the Advanced Encryption Standard Algorithm (AES) was proposed by Wang et al. [20]. This approach aimed to foster improved decision-making concerning cloud storage for users. However, this model did not address key security concerns, including insider attacks, colluding attacks, data integrity issues, potential data intruders, and the threat of malicious files.

Alzain et al. [8] introduced a visionary blueprint for secure storage and file sharing in cloud environments, hinged on a Symmetric Searchable Encryption (SSE) scheme [21–23]. This framework enables users of an electronic healthcare system to securely store their medical data in an encrypted format and perform searches on the data without needing to decrypt it first. Despite the scheme facilitating secure sharing, it lacks efficiency and flexibility due to its non-reliance on policies. Additionally, while the authors broached the subject of access revocation, they fell short of providing a comprehensive and effective answer. Consequently, the architecture is deemed ineffective for sharing substantial volumes of data among various users.

Bhatt et al. [24] analyzes architectural patterns, data governance strategies, and security measures necessary to ensure data integrity, availability, and confidentiality. Through literature review, case studies, and experimental evaluations, the authors propose a novel framework, the Multi-Cloud Data Ecosystem Architecture (MCDEA), designed to address these challenges. Their findings demonstrate that the MCDEA framework can scale to manage large volumes of data, ensure robust security, and optimize resource utilization. However, the paper acknowledges limitations, including the need for further exploration of advanced optimization techniques, the integration of emerging technologies like serverless computing and AI-driven automation, and the adaptation of the framework to handle new multi-cloud complexities as the field evolves.

Ali et al. [25] presented a novel hybrid deep learning model that integrates homomorphic encryption (HE) with a consortium blockchain to enhance the security of Electronic Medical Records (EMRs) in the Industrial Internet of Medical Things (IIoMT). The proposed model improves privacy, security, and efficiency while reducing latency compared to conventional approaches. Additionally, the integration of HE within the IIoMT system strengthens resistance against collusion and phishing attacks. Another work by them introduces a novel Group Theory-Based Binary Spring Search Algorithm integrated with a Hybrid Deep Neural Network to enhance security and efficiency in healthcare systems. Additionally, it presents a secure patient healthcare data access scheme that leverages blockchain and trust chain technology, addressing vulnerabilities in existing frameworks and ensuring secure access to patient health records [26].

Hybrid and Secure Data Sharing Architecture (HSDSA), a privacy-preserving framework for secure data storage in multi-cloud environments combines cryptographic techniques to ensure user control over data generation and decryption, eliminating reliance on a trusted authority. Hajlaoui et al. [27] addresses security, privacy, and data integrity while maintaining reasonable upload and download delays. Experimental evaluations using Cloudera demonstrate HSDSA's efficiency in comparison to existing systems. However, the framework's real-world deployment may face challenges such as data transfer costs, storage fees, network bandwidth, and compatibility with cloud providers' specific security features, which require careful consideration for optimal performance.

Vaidya et al. [17] unveiled a robust structure aimed at fostering secure data interchange within a multi-cloud architecture. This system leans on the use of cryptographic processes, data segmentation, and encryption protocols for public cloud data storage, with metadata (including file partitioning, dissemination information, etc.) conserved on a private cloud database. However, the technique did not incorporate file indexing, leading to an inconvenient retrieval process that necessitates the recipient to engage all slices for file decoding and reconstruction. Further, the research did not sufficiently address key management and distribution, leaving the private cloud database exposed to potential internal threats. Also, the lack of task automation within this structure could potentially decrease its total effectiveness.

A study by [28] proposes Secured multi-cloud information storage framework (SMISF), using ECC and BE for encryption and data chunking. It enhances privacy, prevents unauthorized access, reduces costs, and improves

Table 1 Comparison of various multi-cloud systems

Author	Summary	Strengths	Limitations
Alzain et al. [8]	Surveys the shift from single cloud to multi-cloud (or inter cloud) environments. Proposes the use of secret sharing algorithms to enhance data integrity and protection against intrusion	Improved service availability and resistance to insider threats	Lacking empirical validation, performance overhead, and cost implications
Bessani et al. [23]	Introduces DepSky, ai-cloud-of-clouds storage system designed to enhance the availability, integrity, and confidentiality of critical data stored in the cloud. Mitigates the risks associated with relying on a single cloud through extensive experiments involving four cloud providers and globally distributed clients via PlanetLab	Robust and layered approach to cloud data protection, and reduced redundancy without sacrificing reliability	Increased complexity
Wang et al. [20]	Introduces a certificateless proxy re-encryption (CL-PRE) scheme for secure data sharing in cloud computing environments. Data owners encrypt their data before outsourcing, and a semi-trusted proxy can re-encrypt the data for other users without learning the content	Avoids the need for public key certificates and simplifying key management	Need for robust testing in real-world applications
Fabian et al. [29]	Architecture designed to enhance secure and private inter-organizational data sharing in healthcare using cloud computing. Leverages attribute-based encryption and cryptographic secret sharing to safeguard patient information in semi-trusted cloud environments	Distribute data securely across multiple clouds, limits single points of failure	Scalability issue in key management, single point of failure
Vaidya et al. [17]	Proposes a middleware-based unification framework that creates a Virtual Storage Area Network (VSAN) by integrating multiple Infrastructure-as-a-Service (IaaS) cloud layers, utilizing popular storage services like Dropbox, Box.net, and OneDrive	Enhance data confidentiality and resilience against breaches, simplifies user interactions with multiple cloud platforms	The reliance on public storage APIs could pose limitations due to varying service restrictions, rate limits
Viswanath et al. [30]	Combines AES and the Feistel network to create a hybrid encryption algorithm aimed at enhancing the security of big data before it is stored across multiple cloud services. Includes processes such as data uploading, slicing, indexing, encryption, distribution, decryption, retrieval, and merging to ensure secure storage and retrieval of data	Robust encryption mechanism and enhanced security of the data	May slow down the encryption process and not address all possible security threats
Bhatt et al. [24]	Introduces the Multi-Cloud Data Ecosystem Architecture (MCDEA) that addresses the increasing complexity of managing data in multi-cloud environments. MCDEA supports large-scale data handling, enhances security compliance, optimizes resource use, and enables privacy-preserving collaboration	Scales efficiently across various cloud platforms while maintaining strong security and compliance. Facilitates optimized resource distribution and enables privacy-preserving data analysis	Interoperability between heterogeneous cloud services remains a challenge
Hajloui et al. [27]	Introduces the Hybrid and Secure Data Sharing Architecture (HSDSA) to enhance data security in multi-cloud environments. Combines cryptographic techniques to give users full control over data encryption and decryption, removing the need for a centralized authority and improving trust in cloud systems	Empowering users to manage their own encryption and decryption processes without relying on third-party authorities. Also reducing vulnerability to single points of failure	Introduces computational complexity

performance, giving users control while shielding data from providers and authorities. However, SMISF framework faces scalability, performance, security vulnerabilities, regulatory compliance challenges, user control limitations, and dependence on multiple providers, impacting its effectiveness and reliability in practical applications.

While numerous strategies have been proposed, none have effectively implemented a sound architectural framework or working protocol for secure data sharing using multi-cloud services. Current methodologies do not guarantee file slicing, encryption, decryption, or retrieval processes. They also fail to address important issues including secure key management while sharing data across several cloud storage platforms, conflict resolution during file merging in retrieval, protection from malicious files, defense against colluding providers and insider assaults, centralized data, and defense against harmful files. If encryption occurs before cutting, it becomes difficult to securely upload very big files, frequently resulting in significant delays for users. To address these challenges, this paper introduces an efficient architectural framework employing a standardized algorithm to bolster secure data sharing via index-based cryptographic data slicing. Furthermore, our proposed framework prioritizes data protection against malicious insiders and files during the uploading and downloading process.

3 System model, assumptions, and security goals

3.1 System model

The system model used in this paper is presented in Fig. 1. The proposed CloudLock framework consists of multi-cloud data storage, a Key Dissemination System (KDS), data owners, Authenticated users, and Secure Cloud Storage Enabler (SCSE) unit.

- Multi-cloud storage: The multi-cloud storage system consists of multiple cloud services. It provides space for data owners to keep the encrypted data that consumers may access. It is incharge of letting verified users obtain their data after receiving a request. The multi-cloud storage system integrates Dropbox, Google Drive, and Mega. Data owners upload encrypted files via a secure interface; custom connectors distribute data fragments across providers. Verified consumers authenticate and securely retrieve reassembled files, ensuring high availability, redundancy, and robust security while minimizing risks through client-side encryption and distributed storage.
- KDS: It seeks to offer secure data exchange. The major objective of this unit is to enable secret key sharing among the SCSE unit and the authenticated user. It allows sharing the key among the authenticated user and SCSE unit without actually sharing their private keys.
- Data owners: They are the individuals or the entities or the organizations who own the data. They usually have large amounts of private data which is difficult to store on their local machines and has to be kept confidential.
- Authenticated users: They are the entities who wish to access the data and derive some knowledge from it. These users are authenticated before being given access to the SCSE Unit.
- SCSE Unit: SCSE unit is the sandwich between the multi-cloud framework and data owners or authentic users. It provides the functionalities of data slicing, data distribution, data encryption, data uploading, data allocation, and data decryption.

3.2 Assumptions

The assumptions considered in this work are:

1. All users connected to the system are assumed to be authenticated and trusted, ensuring that they will not

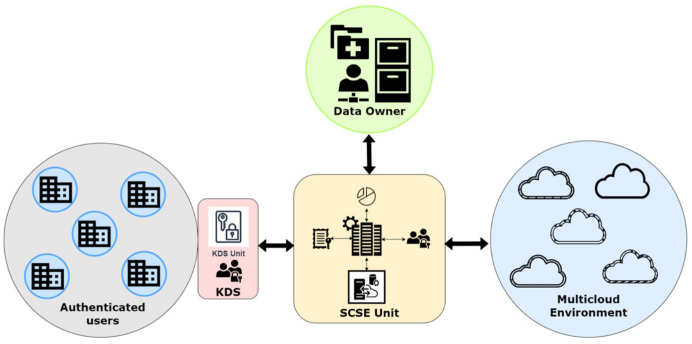


Fig. 1 Proposed system framework

access other users' data without proper authorization. This assumption of authenticated and trusted users serves as a foundational principle for secure systems, facilitating secure information sharing and fostering user trust. By assuming user authentication, this work establishes the following underlying premises:

- Robust authentication mechanisms: The system implements reliable authentication protocols, such as username/password combinations, biometric authentication, or two-factor authentication, to verify the identities of users.
- Authorization controls: The system enforces strict access control policies, defining user permissions and granting access only to authorized individuals or designated user roles.
- Trusted user behavior: Users are expected to adhere to ethical practices and legal requirements, refraining from unauthorized access or misuse of other users' data.
- Absence of malicious intent: The assumption assumes that users connected to the system have no intention to compromise the security of the system or engage in unauthorized activities.

1. All data owners connecting with the system are assumed to be legitimate and do not intend to harm the system. Furthermore, it is assumed that the data owners are only connected to specific users, ensuring that their data will not be accessed or used by unauthorized individuals. This assumption serves as a fundamental premise for this work, establishing the following underlying conditions:

- Legitimate data owners: The work assumes that all data owners connecting to the system are authentic and have lawful ownership or authorized access to the data they possess.
- Absence of malicious intent: It is assumed that the data owners have no intention to compromise the system's security, manipulate or corrupt data, or engage in activities that could harm the system or other users.
- Controlled data access: The assumption encompasses the premise that the system implements appropriate access controls and mechanisms to ensure that each data owner is only connected to specific users with authorized access, thereby preventing unauthorized data sharing or usage.
- Data privacy and confidentiality: The assumption implies that the system upholds strict data privacy measures, safeguarding the data owners' information from unauthorized access and ensuring that it remains confidential and protected from unauthorized usage.

1. The SCSE Unit, as a third-party component, is designed to be highly secure against known threats, and it is presumed that no hacker can gain unauthorized access to its system or compromise its security measures. This assumption establishes the following underlying conditions related to the SCSE unit:

- High-security standards: The SCSE unit is assumed to be developed and maintained with robust security measures, following industry best practices and encryption standards to guarantee the integrity and confidentiality of data handled.
- Resilient infrastructure: It is assumed that the SCSE unit operates within a secure and well-designed infrastructure, including firewalls, intrusion detection systems, and other protective measures, making it highly resistant to unauthorized access attempts.
- Skilled security professionals: The SCSE unit is assumed to be staffed by experienced and knowledgeable security professionals who actively monitor and respond to potential security threats, ensuring the unit is ongoing security posture.
- Thorough security testing: Before deployment, the SCSE unit is put through a thorough security testing process that includes vulnerability assessments, penetration testing, and code reviews.

3.3 Security goals

- Confidentiality of data: During the uploading and downloading of the data, the outsourced data must be protected from eavesdropping threats.
- Verification: Any user's validity to access the data must be verified, and any request applied by the authenticated user for plaintext ($text_{plain}$) must be verified and serviced accordingly.
- Resistance to attacks: Proposed framework should be capable of mitigating possible attacks (cheating, collusion, forgery, and so on) initiated by misbehaving users and adversaries.

4 Preliminaries

4.1 Abbreviations

Table 2 shows the list of abbreviations used throughout the paper.

Table 2 List of abbreviations

Abbreviation	Full form
3DES/triple DES	Triple data encryption standard
AEAD	Authenticated encryption with associated data
AES	Advanced encryption standard
ASD	Associated data
ChaPoly	ChaCha20-Poly1305
CSV	Comma-separated values file
Ct	Ciphertext
CU	Crypter unit
DES	Data encryption standard
DOC	Document
DSDU	Data slicing & Distribution unit
DT	Decryption time
ECDH	Elliptic Curve Diffie-Hellman
ET	Encryption Time
HTML	HyperText Markup language
IND-CPA	Indistinguishability under Chosen plaintext attack
INT-CTXT	Integrity of ciphertexts
KDS	Key dissemination system
Nc	Nonce
PDF	Portable document format
PM	Padded message
RTF	Rich text format
SC	Stream cipher
SCSE	Secure cloud storage encryption
TAT	Turn around time
Tg	Tag
TLS	Transport layer security
TXT	Plain text file

4.2 Methodology to share data secretly

Since its inception in 1979 by Shamir [31] and Blakley [32], secret-sharing systems have been intensively researched [33, 34] follows. The unlocking of a secret is contingent upon a user's ability to collaborate with no less than u–1 additional user, leveraging the information disseminated by the dealer. In this context, u, subject to the condition u $\leq$ n, represents a preset parameter, with 'n' denoting the complete user count. The secret that the dealer and the users must disclose is $\Re \in GF(p_1)$, where $p_1 >$ N. Each user A_i has a secret key $k_i \in GF(p_1)$ that only A_i and the dealer know. The dealer works in two stages. It begins by constructing the polynomial function F(x) of degree u–1 illustrated in:

$$F(x) = \Re + \sum_{i=1}^{u-1} v_i x_i,$$

by randomly selecting each i i.i.d. from GF(p_1) with a uniform distribution. It should be noted that all (addition and multiplication) operations in the above equation are modular arithmetic (defined over GF(p_1)) rather than real arithmetic. Furthermore, F(x) incorporates a constant factor s, which leads to $\Re$ equating to F(0). During the subsequent phase, the dealer disperses a divided secret, $\Re_i$ = F(x_i) to each A_i. In this equation, x_i signifies a random number chosen by A_i for disseminating the secret $\Re$. This number is relayed to the dealer via a secure channel safeguarded by the mutually held key k_i.

Subsequently, we will illustrate how u or a higher number of users can collectively recapture $\Re$ by sharing the secrets they have obtained from the dealer. Let $A_1, \ldots, A_u$ be the collaborating users without losing generality. These u users may deduce the secret $\Re$ = F(0) from $\Re_1$ = F(x_1),..., $\Re_t$ = F(x_t) by doing the following computation:

$$\Re = F(0) = \sum_{i=1}^{u} (\Re_i . \Pi_{j \in |1,n|, j \neq i} (0 - x_J)/(x_i - x_j))$$

It is worth noting that in the above equation, the cumulative product is effectively the Lagrange coefficient. Based on the definition of F(x), the accuracy of the above equation may be easily checked. When recovering the secret $\Re$ in secret sharing, a user may cheat. For example, a user A_i may enter an erroneous $\Re_i$, causing s recovery to fail. To address this issue, verifiable secret-sharing techniques [35, 36] have been suggested, which are mostly based on the RSA cryptosystem, which has a large computational cost.

4.3 Authenticated encryption with associated data (AEAD) syntax

The following three algorithms make up a nonce-based authenticated encryption system with accompanying data:

- The key generation algorithm produces a secret key $\Re$ with no input. To express the key space related to the key generation technique, we overloaded $\Re$.
- The encryption method, Encrypt: $\Re$ * Nc * ASD * $text_{plain}$ $\rightarrow$ Ct takes as inputs a secret key $\Re$, a nonce Nc, associated data ASD, and a message $text_{plain}$ and outputs a ciphertext Ct. Encryption provides a constant expansion, which means that the expansion $|Encrypt(\Re, Nc, ASD, text_{plain})| - |text_{plain}|$ is constant for every $(\Re, Nc, ASD, text_{plain})$.
- The decryption method, Decrypt: $\Re$ * Nc * ASD * Ct $\rightarrow$ $text_{plain}$, takes a secret key $\Re$, a nonce Nc, connected data ASD, and a ciphertext Ct as inputs, and delivers either a message $text_{plain}$ or an error as the output.

The relevant set $\Re$, Nc, ASD, $text_{plain}$, and C are respectively referred to as the key space, nonce space, associated data space, plaintext, and ciphertext. For any ($\Re$, Nc, ASD,

$text_{plain}$), it must hold that if Ct ← Encrypt(ℜ, Nc, ASD, $text_{plain}$), then $text_{plain}$ ← Decrypt(ℜ, Nc, ASD, Ct). This is the accuracy requirement that we place on every nonce-based AEAD.

4.4 ChaCha20

The Salsa stream cipher has been improved by Bernstein into the ChaCha20 stream cipher. An arbitrary-length message $text_{plain}$ (or ciphertext Ct) is encrypted (or decrypted) using a 256-bit secret key ℜ and a 96-bit nonce Nc. It creates a pseudorandom keystream that is XORed to the message, as with any stream cipher. Through the CC block function of the ChaCha20 block algorithm, the keystream is produced in blocks of 512 bits. The 32-bit block counter i and the 96-bit nonce Nc make up the input to the CC block function, which is keyed with ℜ. In this approach, it is used as a pseudorandom function, although in reality, it is a 512-bit permutation set up in a Davies-Meyer fashion. For creating the input for the ChaCha20 permutation, the key, counter, and nonce are specifically fused, preceded by a constant, and subsequently incorporated once again into the output of the permutation through a modulo 232 addition, conducted on a word-by-word basis.

4.5 Poly1305

Bernstein also created the Poly1305 one-time MAC algorithm. It requires a key made up of two strings ($\Im$, ð) totaling 128 bits. Its security is based on Bernstein's demonstration that the hash function V is almost Δ universal. Theorem 2, which is replicated below for completeness, states the security of the hash function V and provides a definition of it.

4.6 Theorems

1. Δ-Universal Hash Functions-Consider V: $KS * DM \rightarrow \{0,1\}^t$ to be a set of keyed hash functions with key space KS, domain DM, and a digest space of $\{0,1\}^t$, for a positive integer t. This study encompasses hash function families operating over both string sets and pairs of strings. When $DM = \{0,1\}^*$, for any given positive real number c, we claim that $\overline{V}$ is c-nearly Δ universal if, for all separate M, M' $\in \{0,1\}^*$ and ∀ z $\in \{0,1\}^t$, the following condition holds true:

$$Pr_{r\leftarrow\$KS}[\overline{V}_r(text_{plain}) = \overline{V}_r(text'_{plain})\overset{(t)}{+}z] \leq \frac{c.max(|text_{plain}|_t, |text'_{plain}|_t)}{2^t} \tag{1}$$

Alternatively, when $DM = \{0,1\}^* \times \{0,1\}^*$, for any distinct $(ASD, Ct), (ASD', Ct') \in \{0,1\}^* \times \{0,1\}^*$ and all $z \in \{0,1\}^t$, we require that:

$$Pr_{r\leftarrow\$KS}[\overline{V}_r(ASD, Ct) = \overline{V}_r(ASD', Ct')\overset{(t)}{+}z] \leq \frac{c.max(|ASD|_t + |Ct|_t, |AD'|_t + |Ct'|_t)}{2^t} \tag{2}$$

2. The Hash Function $\overline{V}$ in Poly1305- Let us consider l as a multiple of 8 that is greater than zero, p as a prime that is larger than or equal to 2^{l+1}, r as a string of l-bits, and $text_{plain}$ as any sequence of bytes. We can decompose $text_{plain}$ as

$$text_{plain} = text_{plain_1}|text_{plain_2}|.....|text_{plain_l}$$

where each $|text_{plain_i}|$ equals l for every i less than or equal to l, and 0 is less than or equal to $|text_{plain_l}|$ and at most l. Hence, we represent $\overline{V_r}(text_{plain})$ as the string of t-bits corresponding to:

$$(I_1 \cdot x^l + I_2 \cdot x^{l-1} + ... + I_l \cdot x^1 \ mod\ p)\ mod\ 2^l, \tag{3}$$

Here, I_i signifies the integer interpretation of the (l+1)-bit string concatenated with $text_{plain_i}$ and 1. Moreover, x represents the integer interpretation of r, post nullifying 22-bit positions (Clamping).

3. $\overline{V}$ is A Δ U- Let $\overline{c} = 2^{25}$, then for any l-bit string s and any pair of distinct byte strings ($text_{plain}$, $text_{plain}$'), it holds that:

$$Pr_{r\leftarrow\$\{0,1\}^l}[\overline{V}_r(text_{plain}) = \overline{V}_r(text'_{plain})\overset{(l)}{+}s] \leq \frac{c.max(|text_{plain}|_l, |text'_{plain}|_l)}{2^l} \tag{4}$$

The single-use MAC (Poly1305 Mac), utilized in ChaCha20-Poly1305, expands the original Poly1305 algorithm to authenticate two strings, not just one. This enhancement is achieved by enriching the hash function with a suitable encoding that maps the pair of strings into a single one, marking a clear separation between the two strings. Crucially, this encoding must be injective. Definition 3. B outlines the process of building V from $\overline{V}$. In Theorem 3. B, we demonstrate that if V is ϵ-almost Δ-universal for individual strings, then V also retains its ϵ-almost Δ-universal property when dealing with pairs of strings, where ϵ (l) equals ϵ (l + 1).

4. The Hash Function V in Poly1305_Mac- Consider r to be a string of t-bits and $\overline{V}$ as the hash function employed in Poly1305. Consequently, the hash of any pair of byte strings represented as (ASD, Ct), can be deduced as follows:

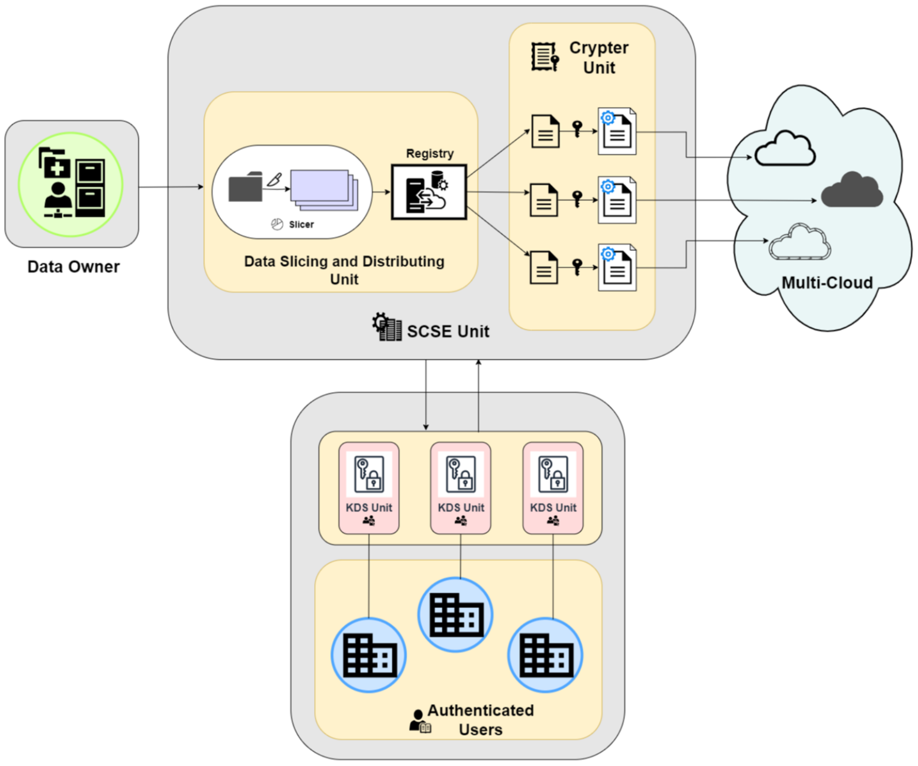


Fig. 2 Proposed CloudLock framework

$$V_r(ASD, Ct) = \overline{V_r}(ASD\|Pad(ASD)\|Ct\| Pad(Ct)\|len(ASD)\|len(Ct)) \tag{5}$$

In this context, Pad(Y) refers to the minimal quantity of zero bytes required so that the bit length of the concatenated $Y|Pad(Y)$ results in a multiple of t. Similarly, len(Y) signifies the representation of the byte length of Y in t/2-bits.

5 The proposed CloudLock framework

Secure data sharing in a multi-cloud environment is a crucial aspect of handling data storage problems. A single cloud could act as a central point of failure in case of attack which leads the hackers or intruders to get access to organizations' data. This also deduces that the data stored in cloud storage may not be safe. So, in order to handle the listed problems, we proposed CloudLock, a reliable framework data storage and sharing schema for a multi-cloud environment. It omits the idea of a central point of failure of a single cloud by slicing the data into sub-parts and storing the sub-parts on multiple clouds. Also, it resolves the issue of data security by using encryption schemes. Proposed CloudLock framework consists of SCSE unit which acts as a middleware between the clouds, authenticated users, and data owners and consists of various units which aid to ease the process of uploading, encryption, decryption, and downloading the data. Figure 2, gives an overall idea of the proposed framework. Formerly, the owner of data stores their files on a multi-cloud platform by sending a request to the SCSE Unit. Its sub-unit named Data Slicing & Distribution Unit (DSDU), first divides the data into multiple parts and sends them to the registry, where the registry acts as a hashing module and provides indexing to the file for merging and storing purposes. Next, these files are transferred to the Crypter unit for encryption and then they are uploaded to multiple cloud platforms. The number of cloud platforms varies from organization to organization depending upon the amount of data and the cost to buy resources. While accessing the file, each authenticated user has to send a request to the SCSE unit for file access. This request turns on the file access mechanism, described in Algorithm 3 which enables the authenticated user to access the data.

A data owner has a data, say F = $\{F_1, F_2, F_3, \ldots\ldots, F_n\}$, which is to be uploaded to the cloud storage Cd = $\{Cd_1, Cd_2, Cd_3, \ldots\ldots, Cd_n\}$. At this stage, the data owner requests the SCSE unit to upload its file on the multi-cloud environment. Initially, the SCSE unit instantiates the DSDU unit to commence the data slicing and distribution phase. Next, the Registry allocates the data to multiple

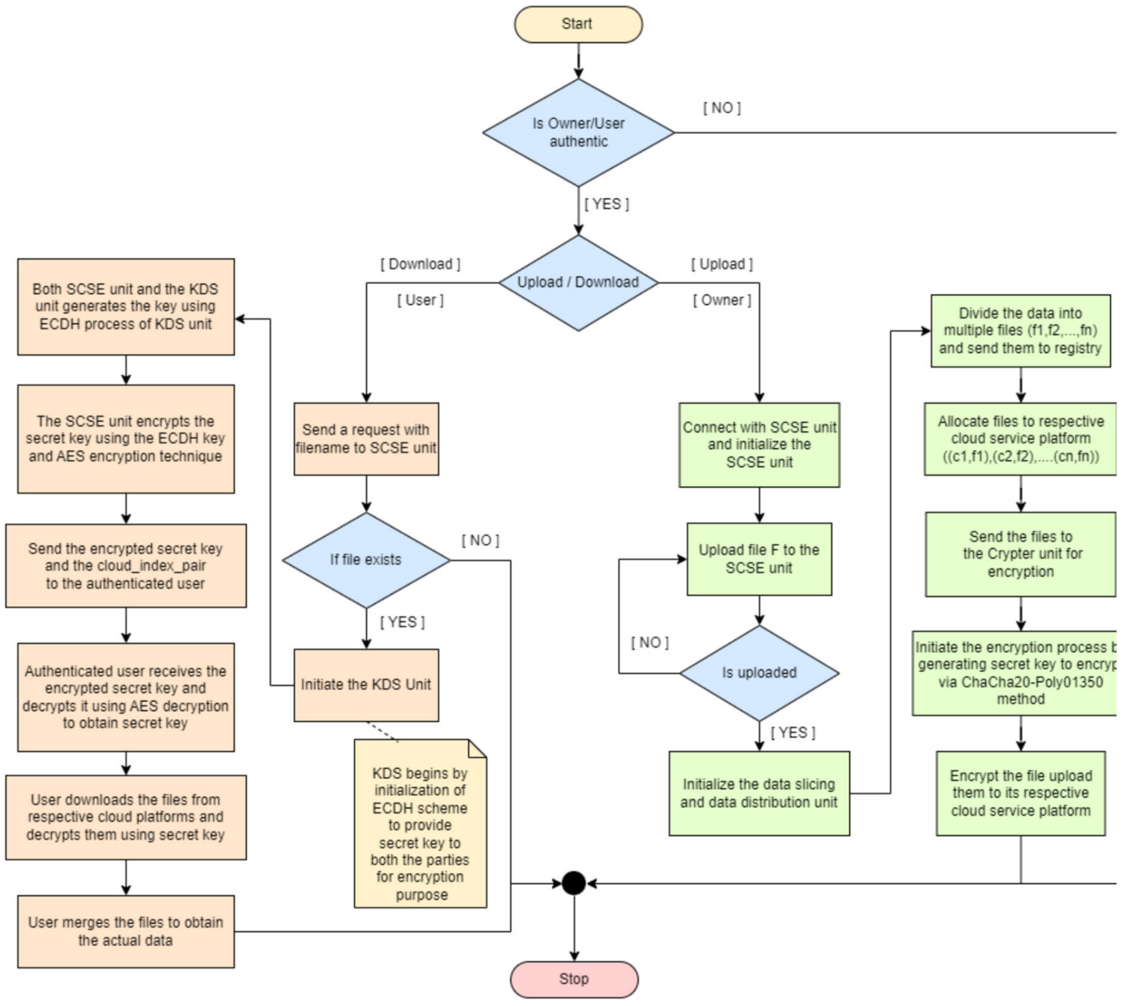

Start
Is Owner/User authentic
[NO]
[YES]
Upload / Download
[Download]
[User]
[Upload]
[Owner]
Both SCSE unit and the KDS unit generates the key using ECDH process of KDS unit
The SCSE unit encrypts the secret key using the ECDH key and AES encryption technique
Send the encrypted secret key and the cloud_index_pair to the authenticated user
Authenticated user receives the encrypted secret key and decrypts it using AES decryption to obtain secret key
User downloads the files from respective cloud platforms and decrypts them using secret key
User merges the files to obtain the actual data
Send a request with filename to SCSE unit
If file exists
[NO]
[YES]
Initiate the KDS Unit
KDS begins by initialization of ECDH scheme to provide secret key to both the parties for encryption purpose
Connect with SCSE unit and initialize the SCSE unit
Upload file F to the SCSE unit
Is uploaded
[NO]
[YES]
Initialize the data slicing and data distribution unit
Divide the data into multiple files (f1,f2,...,fn) and send them to registry
Allocate files to respective cloud service platform ((c1,f1),(c2,f2),....(cn,fn))
Send the files to the Crypter unit for encryption
Initiate the encryption process b generating secret key to encryp via ChaCha20-Poly01350 method
Encrypt the file upload them to its respective cloud service platform
Stop

Fig. 3 Workflow of the proposed CloudLock framework

clouds which are stored in the clouds after passing through the Crypter unit. Figure 3 describes the workflow of the proposed framework. The process begins with user authentication; if the user is not authentic, the process is terminated. Upon successful authentication, the user can either upload or download files. In the upload process, the data owner connects with the SCSE unit and uploads the file. The system then divides the file into multiple segments, registers them, and allocates each segment to different cloud platforms. Before storage, the data undergoes encryption via the ChaCha20-Poly1305 encryption method, ensuring confidentiality. Each encrypted file segment is then uploaded to its respective cloud storage platform. In the download process, an authenticated user requests a file by sending its filename to the SCSE unit. If the file exists, the KDS is initiated to generate a secret key using the ECDH scheme. The SCSE unit encrypts the secret key using AES encryption and transmits it securely to the user. The user decrypts this secret key and downloads the encrypted file segments from the cloud platforms. Using the obtained key, the user decrypts and merges the file segments to reconstruct the original data. This approach enhances data security by utilizing encryption techniques and distributing storage across multiple cloud platforms.

5.1 SCSE Unit

SCSE Unit is the sandwich between the multi-cloud framework and data owners or authentic users. It provides the functionalities of data slicing, data distribution, data encryption, data uploading, data allocation, and data decryption. Data Slicing is the fragmentation of data into subparts which is followed by data distribution. It consists of distributing data to cloud services and storing the allocated Cloud_Index_pairs in Registry. Additionally, it constitutes of Crypter Unit whose functionality is to encrypt & decrypt the data using ChaCha20-Poly1305 encryption and decryption respectively. It also empowers to encrypt the secret key, $\Re$, used for file encryption before sending it to the authenticated user.

Algorithm 1 ChaCha20-Poly1305 AEAD scheme

```
Input: $\Re \in \{0,1\}^{256}$, Nc $\in \{0,1\}^{96}$, ASD $\in \{0,1\}^*$, $text_{plain} \in \{0,1\}^*$ where $\Re$ is
the secret key (32-byte), Nc is nonce (12-byte), and ASD is a variable-length
associate data.
Output: Ct $\in \{0,1\}^{\|text_{plain}\|}$ , Tg $\in \{0,1\}^{128}$
    ChaCha20-Stream Cipher (SC)
1:  for j = 0 to $\lceil |text_{plain}|/512 \rceil - 1$ do
2:  /*Initialization state*/
3:  St[0] ← 0x61707865 , St[1] ← 0x3320646e
    St[2] ← 0x79622d32, St[3] ← 0x6b206574
        where: 0x61707865, 0x3320646e, 0x79622d32,
        0x6b206574 are constants
        St = Initial State (4 * 4 matrix)
        St[e] = element at position e (each St[e] is represented by 32-bit word)
4:  St[4..11] ← $\Re$ { Set key }
5:  St[12] ← j {Set counter}
6:  St[13..15] ← Nc {Set nonce}
7:  St' ← St
8:  for $\eta$ ← 0 to 9 do
        /*Column round*/
        St[0,4,8,12] ← $Q_r(St[0], St[4], St[8], St[12])$
        St[1,5,9,13] ← $Q_r(St[1], St[5], St[9], St[13])$
        St[2,6,10,14] ← $Q_r(St[2], St[6], St[10], St[14])$
        St[3,7,11,15] ← $Q_r(St[3], St[7], St[11], St[15])$
        /*Diagonal Round*/
        St[0,5,10,15] ← $Q_r(St[0], St[5], St[10], St[15])$
        St[1,6,11,12] ← $Q_r(St[1], St[6], St[11], St[12])$
        St[2,7,8,13] ← $Q_r(St[2], St[7], St[8], St[13])$
        St[3,4,9,14] ← $Q_r(St[3], St[4], St[9], St[14])$
    end for
    $Note_1$: $Q_r$ = Quarter function
    It is defined as :
    (a,b,c,d) = $Q_r$(p,q,r,s), where:
    a=(p ⊞ q) ⊞ (q ⨁ (r ⊞ (s ⨁ (p ⊞ q) <<<16))<<<12)
    d=((s ⨁ (p ⊞ q)<<<16) ⨁ a) <<< 8
    c=(r ⊞ (s ⨁ (p ⊞ q) <<< 16)) ⊞ d
    b=(((q⨁(r ⊞ (s⨁(p ⊞ q)<<<16))) <<<12)⨁c)<<<7
    and ⊞ = integer addition module
    ⨁ = XOR operation
9:  $\Re_j^{Nc}$ ← St ⊞ St'
    { ∀ 0 ≤ e ≤ 15 : St[e] ⊞ St'[e]}
10: $Ct_j \leftarrow PT_j \bigoplus k_j^N c$ {Encrypt}
11: end for
12: Save Ct $\in \{0,1\}^{\|text_{plain}\|}$
13: PM ← Pad(ASD) ‖ Pad(Ct) ‖ Le(ASD) ‖ Le(Ct) {Pad Message(PM)}
    $Note_2$: L : $\{0,1\}^* \to \{0,1\}^{64}$
    Le returns the length of input as 64-bit little-endian integers
    Pad: $\{0,1\}^{8b} \to \{0,1\}^{8(b+\delta)}$        17
    Pad returns the b-byte input padded with $\delta$ = (16-b)mod 16 zero bytes
14: ($\Im$, ð)← ChaCha($\Re$,Nc,0) {Compute one-time key}
    Poly-1305
15: $(h_1, h_2, ....h_t)$ ← Pad1305(H) {Chop H into t chunks}
    $Note_3$: H ⊂ $text_{plain}$, where H is a b-byte message
    H $\in \{0,1\}^{8b}$
    Pad1305 : $\{0,1\}^{8b} \to \{0,1\}^{136t}$
    where: t = ⌈l/16⌉ 17-byte chunks $(h_c)_{1 \le c \le t}$ where each 16-byte block is padded
    with byte 0x01 and last block is padded with byte 0x01 & (16-b) mod 16 zero bytes
16: $\overline{\Im}$ ← Clamp ($\Im$) { Clear bits of $\Im$}
    $Note_4$: Clamp: $\{0,1\}^{128} \to \{0,1\}^{128}$
    $\overline{\Im} = \Im_0 + \Im_1 + \Im_2 + \Im_3$,
    $\Im_0 \in \{0, 1, 2, ...., 2^{28} - 1\}$,
    $\Im_1/2^{32} \in \{0, 4, 8, ...., 2^{28} - 4\}$,
    $\Im_2/2^{64} \in \{0, 4, 8, ...., 2^{28} - 4\}$ &,
    $\Im_3/2^{96} \in \{0, 4, 8, 12, ...., 2^{28} - 4\}$
17: v ← 0
18: for j ← 0 to (t -1) do {$Evaluate\ Polynomial$}
        v ← v + ($h_{j+1}$ . $\overline{\Im}^{t-j}$ mod $2^{130}$) - 5
19: end for
20: Evaluate Tg,
        Tg ← v + ð mod $2^{128}$ {$Generate\ Tag$}
    $Note_5$: Tg is overall evaluated by using the equation:
    Tg = ($\sum_{j=1}^{t} .\overline{\Im}^{t-j+1}$ mod $2^{130}$ - 5) + (ð mod $2^{128}$)
21: Save Tg
22: return (Ct, Tg)
```

5.1.1 Data slicing and distribution unit

This unit deals with data division and data distribution. Its subcomponents are described as follows:

Data Slicing: This is the first and foremost step prior to data uploading. Here, the data file, say F, is divided into multiple files

$$F = \{f_1, f_2, f_3, \ldots., f_n\} \tag{6}$$

where n varies with the number of clouds chosen for storing the data. It seeks to improve security for owner data on many clouds. This action is crucial because:

1. Distributing over multiple clouds helps to store data easily as compared to the single cloud
2. It avoids any problems related to data size restrictions
3. It allows efficient usage of bandwidth
4. It aids load balancing
5. It facilitates to encryption and decryption process, by considerably decreasing the size of the file to be encrypted or decrypted.

Registry: A registry is the most important component of the SCSE unit as it contains all the metadata and all the symmetric keys $\Re$s'. It is a directory system which stores Cloud_ID and File_Index hash pairs, termed as $\{(Cd_1, f1), (Cd_2, f_2), (Cd_3, f_3), \ldots\ldots, (Cd_n, f_n)\}$. It enables the system to maintain the record of the file F. Also, it stores the private key used to encrypt the data encrypt $\{f_i, \Re\}$. $\Re$ is encrypted after obtaining private key Ⅎ by KDS.

$$\aleph = encrypt(\Re, Ⅎ) \tag{7}$$

This $\Re$ is sent to the authentic user upon request through Key Dissemination System (KDS), send($\aleph$).

5.1.2 Crypter Unit (CU)

This unit deals with the encryption schema. It allows the files to be encrypted using the ChaCha20-Poly1305 encryption. When a file is passed from the Registry then it gets passed on to the Crypter Unit to get encrypted where after its encryption it is uploaded to the cloud service platforms. The basic concept behind the ChaCha20-Poly1305 is described as follows:

- The ChaCha20 is a stream cipher and the Poly1305 authenticator could be combined together to build the ChaCha20-Poly1350 AEAD scheme [37]. Random oracle model proves the IND-CPA & INT-CTXT security of this scheme on the assumption that Poly1350 is a Δ-universal hash function [38].
- The AHED schema requires the input of a 32-byte secret key $\Re$, a 12-byte Nonce Nc, a variable length $text_{plain}$, & a variable length associate data (ASD) & returning Ct and a 16-byte authentication tag Tg. CU: $\{0,1\}^{256} * \{0,1\}^{96} * \{0,1\}^{*} \rightarrow \{0,1\}^{*} * \{0,1\}^{128}$ such that ($\Re$,Nc,$text_{plain}$,ASD) → (Ct,Tg).

Here the encryption is performed using Algorithm 1 while taking (block counter) $\gamma \geq 1$ as $\gamma = 0$ is used to generate the one-time key ($\Im$, ð) & so it cannot be reused to encrypt. It begins by encrypting the plaintext using the ChaCha20 stream cipher: a 256-bit secret key, a 96-bit nonce, and a counter are used to initialize a 4×4 state matrix alongside fixed constant values. The state undergoes 20 rounds of transformation-alternating between column and diagonal rounds via the quarter round function (Q_r) to produce a keystream block. Each plaintext block is then XORed with its corresponding keystream block to yield the ciphertext. Once encryption is complete, the algorithm constructs a padded message by concatenating the padded associated data, the padded ciphertext, and their respective lengths in little-endian format. A one-time key for the Poly1305 MAC is derived from a separate ChaCha20 invocation (with counter zero) and is "clamped" (i.e., certain bits are cleared) to meet specific security requirements. The padded message is then divided into fixed-size chunks, each processed as part of a polynomial evaluation modulo 2^{130}. After evaluating the polynomial-with a final adjustment by adding the clamped one-time key-the algorithm computes a 128-bit authentication tag. The final output consists of the ciphertext and the tag, which together provide authenticated encryption.

In order to decrypt the functionality of the $text_{plain}$, & Ct is reversed and the generated tag must be bitwise compared with the received tag in order to verify the authenticity of data. A concept to be noted is that the authenticator of the proposed framework takes Ct as its input to encrypt as well as to decrypt (Algorithms 1 & 2). The groundwork for ChaCha20-Poly1305 is laid as:

Algorithm 2 ChaCha20-Poly1305 decryption algorithm

```
Input: $\Re \in \{0,1\}^{256}$, Nc $\in \{0,1\}^{96}$, ASD $\in \{0,1\}^*$, Ct $\in \{0,1\}^*$, Tg $\in \{0,1\}^{128}$
Output: Plain Text $text_{plain}$
1: $\Im||\eth \leftarrow$ Poly1305_Key_Gen($\Re$,Nc), step 14 in algorithm 1
2: Tg' $\leftarrow$ Poly1305_MAC(($\Im, \eth$),ASD,Ct), steps{15..21} in algorithm 1
3: If Tg' $\neq$ Tg then return error
4: return ChaCha20($\Re$,Nc,Ct), steps{1..12} in algorithm 1
```

The AHED composition: the encryption and decryption algorithms consist of subparts, ChaCha20, one-time Poly1305, and Poly1305_key_gen which are based on the Δ-universal hash function family V over string pairs. Here one-time key ($\Im, \eth$) is derived again for every encryption process by running the ChaCha20 block function in Poly1305_Key_Gen on $\Re$, Nc, and the counter value zero (saved specifically for this thing). On the contrary, the decryption algorithm works vice versa by computing the one-time key first, then recomputing the MAC tag and checking if the tag is identical to Ct used with ChaCha20 & returns the deciphered text, else returning an error.

Algorithm 3 File access algorithm

```
Input: Initialization request for ECDH
Output: Symmetric Key $\daleth$
   // Set-up phase
1: Select elliptic curve $E_c$
       $E_c$: $y^2 = x^3$ + px + q (mod $p_r$)
       where, $p_r$ = prime number, p & q = coefficients of
       specific curve
2: Pick a base point on $E_c$, say $\eth$ with order m, i.e.,
       $\eth^m$ = e
       where e = identity element
   // Key generation phase
3: Both $Cd_u$ & $A_u$ randomly chooses their private key say $\zeta_k$ & $\xi_k$ respectively
4: Both $Cd_u$ & $A_u$ generates their public key $\wp$ & $\rho$ respectively :
       $\wp = \zeta_k.\eth$
       $\rho = \xi_k.\eth$
   // Key exchange & formation
5: $A_u$ sends $\rho$ to $Cd_u$ through KDU. $Cd_u$ performs $\zeta_k.\rho$ i.e.,
       $\daleth = \zeta_k.\xi_k.\eth$
6: $Cd_u$ sends $\wp$ to $A_u$ through KDU. $A_u$ performs $\xi_k.\wp$ i.e.,
       $\daleth = \xi_k.\zeta_k.\eth$
   Note: This $\daleth$ acts as the symmetric key for encrypting & decrypting secret key $\Re$
```

These mechanisms could handle various security attacks, including man-in-the-middle (MITM) attacks, key interception, and unauthorized access. By utilizing ECDH for key exchange and AES encryption for securing the secret key, the system ensures that only authenticated users can access and decrypt the stored files, enhancing data confidentiality and integrity in cloud environments.

5.2 Key dissemination system

Data Accessing is one of the major aspects dealt with in this paper. Utilizing the KDS, it seeks to offer secure data exchange. The major objective of this unit is to enable secret key sharing among the SCSE unit and the authenticated user as shown in Fig. 4. The process begins when a user requests file access (Step 1). To establish a secure key exchange, both the user and the file owner generate their respective keys using the Elliptic-Curve Diffie-Hellman (ECDH) mechanism (Steps 2–3). The Secure Cloud Storage Encryption (SCSE) unit then encrypts the secret key, which was originally used for file encryption, using the generated ECDH key (Step 4). Next, the SCSE unit transmits the encrypted key along with the cloud index pair to the Key Distribution System (KDS) (Step 5). Upon receiving the encrypted key, the KDS performs decryption to retrieve the original secret key (Step 6). Finally, the authenticated user receives the decrypted secret key, which is essential for file decryption and reconstruction (Step 7). This process ensures a secure exchange of encryption keys, allowing only authenticated users to access and decrypt the requested files.

It allows sharing the key among the authenticated user and SCSE unit without actually sharing their private key through the Elliptic Curve Diffie Hellman (ECDH) mechanism. The final key obtained, $\daleth$ after the ECDH process is used as a symmetric key for encryption of the true secret key, $\Re$ using the AES mechanism. This encrypted key is then sent to the user along with the Cloud_Index_pairs to obtain the original file. Algorithm 3 describes the process to access files from the multi-cloud storage.

5.2.1 Elliptic Curve Diffie Hellman (ECDH) mechanism

The ECDH algorithm is a key exchange protocol that provides a reliable method for two users to share a secret over an unsecured network. It is devised through the mathematical properties of elliptic curves (EC) & is widely used in modern cryptographic systems. It is a modification of the original Diffie-Hellman (DH) algorithm, which operates in the group of integers moduled with a prime. Instead of using integer arithmetic, ECDH employs operations on points of an EC defined over a finite field. This makes the algorithm more efficient and provides a larger level of security for the same key size. Its security relies on the Elliptic Curve Discrete Logarithm Problem (ECDLP), stating that it is computationally infeasible to determine the private key from the public key. The EC's mathematical structure makes solving the ECDLP significantly more challenging compared to the classical discrete logarithm problem used in traditional Diffie-Hellman algorithm. Moreover, ECDH provides several advantages over other key exchange algorithms. Because it provides robust security with comparatively tiny key sizes, it is computationally effective and ideal for devices with limited resources. In addition, ECDH is immune to quantum computer attacks, which are a danger to several conventional encryption techniques.

The ECDLP states that given a point a_1 on an elliptic curve, it is computationally infeasible to determine the integer n such that n . a_1= a_2, where a_2 is another point on the curve. In other words, it is arduous to identify the private key from the public key.

The security of ECDH is influenced by two main factors:

- The size of the finite field and the choice of EC parameters
- Increasing the size of the finite field increases the difficulty of solving the ECDLP.

Algorithm 4 Key dissemination process

```
Theory: Say authenticated user $A_u$ wants the access of data F present in a
multi-cloud storage system $Cd_d$, d ∈ {1..n} where 1 to n are the multiple clouds used
by the Data owners. Thus, to get the encrypted data present in multi-cloud, a secret
key $\Re$ is required to decrypt it. So, before accessing the data stored in multiple
clouds, it uses Key Dissemination System (KDS) to obtain the encrypted key $\aleph$.

Input: File request R(F), where $F$ is file to be accessed
Output: Encrypted secret key $\aleph$ & Cloud_index_pair ($Cd_{id}$,$f_{id}$)
1: $A_u$ sends R(F) to KDS
2: KDS uses F to notify the SCSE unit $Cd_u$ for file F.
3: $Cd_u$ sends acknowledgment & KDS responds it by initiating the key dissemination
   process, mentioned algorithm 4.
4: $Cd_u$ uses ECDH generated key $\daleth$ to encrypt the $\Re$;
       $\aleph$ = encrypt ( $\Re$ , $\daleth$ )
5: $Cd_u$ sends 'f' which contains the encrypted secret key $\aleph$ and the cloud_index_pair
   to KDS:
       f = send ($\aleph$, [$Cd_{id}$, $f_{in}$])
       where $Cd_{id}$ = Cloud_ID, $f_{in}$ = File_index
6: KDS sends the f to the $A_u$;
       $A_u$ = Receive (f)
7: $A_u$ uses its key $\daleth$ to decrypt $\aleph$
       $\Re$ = decrypt ($\aleph$, $\daleth$)
8: $A_u$ uses $\Re$, $Cd_{id}$, $f_{in}$ to merge & obtain required file.
       F = Merge ($\Re$, [$Cd_{id}$, $f_{in}$])
```

6 Results and analysis

6.1 Experimental setup

To evaluate the proposed CloudLock framework, we considered the scenario in which the encrypted data is stored in multi-cloud storage services such as Dropbox, Google Drive, and Mega. Python 3.10 was used to create the encryption and decryption system. The performance of the proposed framework is compared with cutting-edge approaches like Advanced Encryption Standard (AES), DES, Triple DES, and ChaCha20. Turn Around Time (TAT), encryption time, decryption time, and memory usage are the factors used for assessment. To test the suggested technique, the time duration for uploading and downloading from the indicated storage platforms is also determined. All simulation trials are carried out with different files such as Text, PDFs, RTFs, DOCs, HTML, and CSV with varying sizes ranging from 1 to 300 MB. Furthermore, to better comprehend the difference, .CSV and .TXT files with 1GB of data were also used to compare with state-of-the-art methodologies.

6.2 Performance analysis

The performance analysis of the proposed framework is presented in this section based on the encryption time (ET) and decryption time (DT) using Chacha-Poly1305 (ChaPoly)for different file sizes and types. The results are summarized in Table 3. The table provides a comparison of encryption and decryption times for various file sizes and formats.

The results indicate that as the file size increases, the encryption and decryption times tend to increase as well. Moreover, certain file formats, such as HTML, exhibit higher encryption and decryption times compared to others. These findings provide insights into the performance characteristics of the proposed framework for file encryption and decryption, which can be valuable for evaluating its practicality and efficiency in real-world scenarios.

For the TXT files, ET varies between 0.309 s for a 1 MB file and 0.706 s for a 300 MB file. The corresponding DT range from 0.318 to 0.635 s. Whereas for CSV files, the encryption times range from 0.169 to 0.577 s, while the decryption times vary between 0.156 and 0.492 s. The encryption and decryption times for PDF files vary from 0.166 to 0.738 s, and 0.163 to 0.743 s respectively. The encryption and decryption times for PDF files generally increase as the file size increases. The RTF files show encryption times between 0.160 and 0.894 s, and decryption times between 0.162 and 0.661 s.

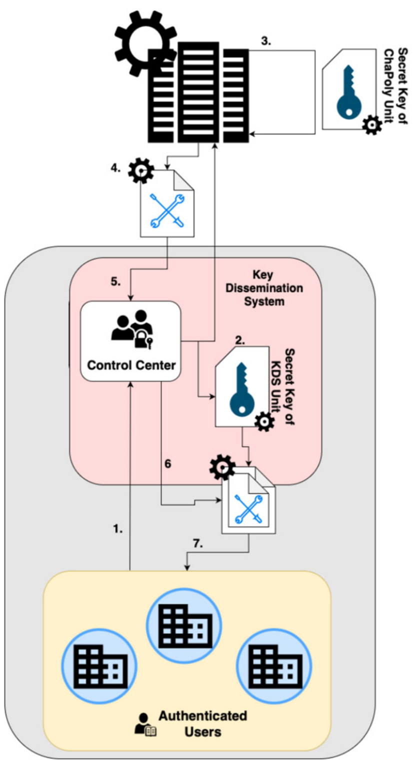


Fig. 4 Key dissemination system ((1) User request for file access, (2–3) User and owner generate their keys using ECDH mechanism, (4) SCSE unit encrypts the secret key used for actual file encryption using ECDH key, (5) SCSE unit send the encrypted key to KDS with cloud index pair, (6) Decryption of received key, (7) Authenticated user received the Secret key used for File decryption)

For XLS files, the encryption times range from 0.180 to 0.732 s, and decryption times vary between 0.184 and 0.640 s. The encryption and decryption times for XLS files exhibit a similar increasing trend with increasing file size. Further, for HTML files, the encryption times range from 0.311 to 1.512 s, while the decryption times vary between 0.291 and 2.122 s. The encryption and decryption times for HTML files display the highest values among all the file types, and they also increase significantly with larger file sizes. It could be noted that generally, the encryption time doubles when the size of the file is doubled regardless of the file type.

Table 4 compares the ET and DT for TXT and CSV files with varying sizes using different encryption algorithms, including ChaPoly used in proposed approach, AES,[1] DES, ChaCha20, and Triple DES.

For .TXT files, the encryption and decryption times vary depending on the file size and encryption technique. For a 1 MB .TXT file, the ChaPoly encryption technique has an encryption time of 0.309 s and a decryption time of 0.318 s. In contrast, AES, DES, ChaCha20, and Triple

[1] To avoid any confusion, here AES is used for Encryption and Decryption of the files not the key.

Table 3 Different file encryption and decryption time comparison of proposed framework

File size (MB)	TXT file		CSV file		PDF file		RTF file		XLS file		HTML file	
	ET	DT	ET	DT	ET	DT	ET	DT	ET	DT	ET	DT
1	0.309	0.318	0.169	0.156	0.166	0.163	0.160	0.162	0.180	0.184	0.311	0.291
10	0.340	0.376	0.184	0.182	0.197	0.174	0.179	0.179	0.184	0.185	0.353	0.346
50	0.361	0.465	0.251	0.251	0.243	0.242	0.257	0.243	0.258	0.243	0.462	0.510
100	0.339	0.326	0.338	0.322	0.356	0.314	0.355	0.341	0.365	0.334	0.699	0.640
200	0.532	0.474	0.551	0.474	0.520	0.503	0.553	0.503	0.585	0.541	1.008	1.638
300	0.706	0.635	0.577	0.492	0.738	0.743	0.894	0.661	0.732	0.640	1.512	2.122

Table 4 Different techniques encryption and decryption time comparison

File type	File size	ChaPoly		AES		DES		ChaCha20		Triple DES	
		ET	DT	ET	DT	ET	DT	ET	DT	ET	DT
.TXT	1 MB	0.309	0.318	0.008	0.002	0.014	0.012	0.003	0.002	0.029	0.025
	10 MB	0.340	0.376	0.021	0.020	0.093	0.086	0.012	0.019	0.240	0.246
	50 MB	0.361	0.465	0.069	0.077	0.424	0.434	0.052	0.058	1.178	1.166
	100 MB	0.339	0.326	0.135	0.136	0.817	0.816	0.102	0.101	2.275	2.278
	200 MB	0.532	0.474	0.236	0.251	1.653	1.639	0.192	0.192	4.507	4.481
	300 MB	0.706	0.635	0.338	0.333	2.479	2.541	0.288	0.309	6.757	6.679
	1GB	2.363	2.094	1.540	1.205	8.422	8.547	1.089	1.046	22.965	23.091
.CSV	1 MB	0.169	0.156	0.005	0.002	0.012	0.015	0.004	0.002	0.029	0.027
	10 MB	0.184	0.182	0.022	0.019	0.092	0.088	0.011	0.011	0.236	0.253
	50 MB	0.251	0.251	0.070	0.071	0.421	0.442	0.056	0.060	1.162	1.165
	100 MB	0.338	0.322	0.130	0.130	0.824	0.837	0.099	0.107	2.299	2.302
	200 MB	0.551	0.474	0.245	0.240	1.657	1.631	0.184	0.195	4.525	4.524
	300 MB	0.577	0.492	0.340	0.346	2.463	2.447	0.283	0.299	6.777	6.735
	1GB	2.447	2.351	1.550	1.482	8.329	9.495	1.104	1.017	22.574	22.636

DES have significantly lower encryption and decryption times, ranging from 0.008 to 0.029 s for encryption and from 0.002 to 0.025 s for decryption. As the file size increases, all encryption and decryption times for .TXT files also increase, but the relative performance between the techniques remains consistent.

Similarly, for .CSV files, the encryption and decryption times vary with file size and encryption technique. For a 1 MB .CSV file, the encryption times range from 0.169 s (ChaPoly) to 0.005 s (AES), while the decryption times range from 0.156 s (ChaPoly) to 0.002 s (AES). As the file size increases, the encryption and decryption times for .CSV files also increase for all encryption techniques, but the relative performance among the techniques remains consistent.

Overall, the results show that hybrid ChaPoly used in proposed CloudLock framework generally has a bit higher encryption and decryption times compared to other encryption techniques across both .TXT and .CSV file types and different file sizes. AES consistently demonstrates the lowest encryption and decryption times among the evaluated techniques for both file types. DES and ChaCha20 exhibit intermediate performance, while Triple DES generally has the highest encryption and decryption times. The higher value for encryption and decryption time of ChaCha20-Poly1305 could be given to the fact that this technique not only provides encryption-decryption schema but also uses the Poly1305 authenticator, which is unique to the fact that none of the other existing techniques possess. Moreover, this authenticator adds more security to the ChaCha20-Poly1305 scheme as compared with other encryption techniques. Additionally, the memory comparison between ChaCha20-Poly1305 with other encryption techniques as shown in Figs. 5 and 6 indicates that our proposed framework uses lesser memory than AES which helps us to implement it in case of applications or systems where more priority is given to the memory as compared to the time.

Unlike conventional encryption methods that rely on a single cryptographic technique, the hybrid system

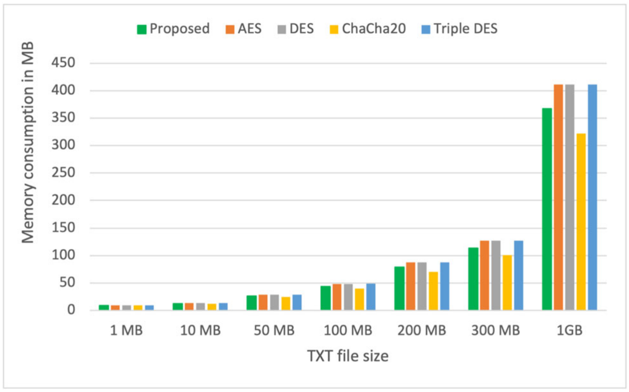


Fig. 5 Memory consumption of different encryption techniques for TXT data

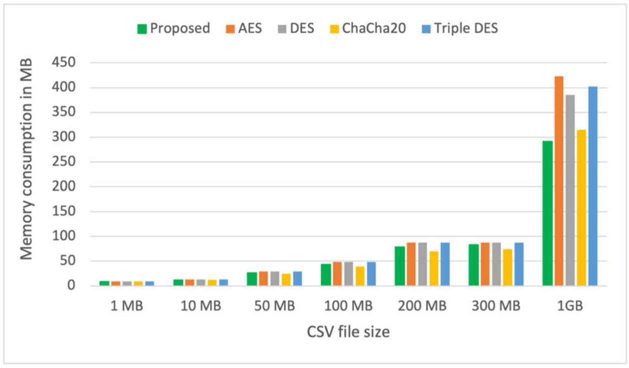


Fig. 6 Memory consumption of different encryption techniques for CSV data

dynamically adapts to different cloud security frameworks, ensuring compatibility across multiple providers. Furthermore, by using asymmetric cryptography solely for key exchange and symmetric encryption for data protection, the system optimizes performance, reducing computational overhead and making it well-suited for large-scale data sharing. This approach effectively prevents unauthorized access by ensuring that even if an attacker gains access to cloud storage, they cannot decrypt the data without the private key, which is securely distributed through the Key Dissemination System (KDS).

This schema could be implemented in resource-restrained scenarios where we could only use a limited number of resources. It is worth noting that the encryption and decryption times increase as the file size increases for all encryption techniques and file types. This is expected as larger files require more computational resources for encryption and decryption processes. So, ChaCha20-Poly1305 would work in a more efficient way than other existing techniques as these multi-cloud systems have to generally deal with large data files.

Figure 7 indicates the memory consumption of different data types with the CloudLock in MBs. A general

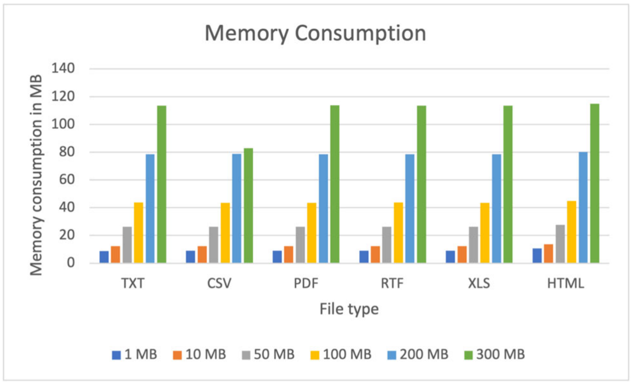


Fig. 7 Memory consumption of different data types with proposed CloudLock framework

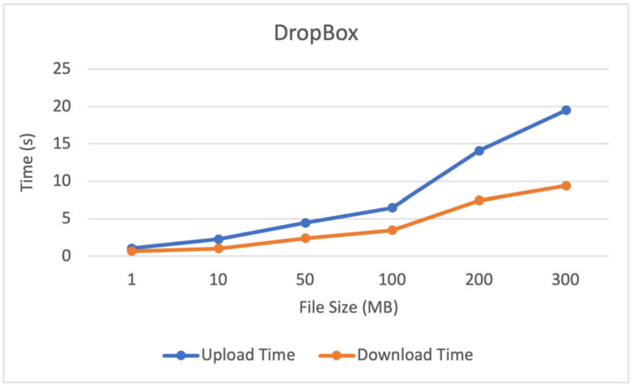


Fig. 8 Upload and download time for Dropbox with proposed framework on CSV data

increasing trend could be observed from the figure indicating that memory consumption increases with an increase in the size of the file. Additionally, it could also be observed that there is a slight variation between multiple files indicating that memory consumption does not vary with file type but is influenced by file size.

Figures 8, 9, and 10 depicts the upload and download time for multiple service platforms like Dropbox, Google Drive, and Mega. It is seen that the upload time is similar to the download time up to 10MB and then a significant gap emerges between them with an increase in the file size.

In the case of Dropbox, the upload time varies from 1.05133 to 19.4898 s whereas the download time ranges from 0.663 to 9.379 s for file sizes 1MB to 300MB.

In the case of Google Drive, the upload time ranges from 1.256 to 17.808 s whereas the download time changes from 0.936 to 9.311 s for file sizes 1MB to 300MB.

In the case of Mega, the upload time and download time for 1MB file is similar but a large difference is observed between them when the file size is increased to 300 with values ranging from 0.916 to 53.141 s to upload and 0.476 to 8.690 s for download.

Figure 11 describes the Turn Around Time (TAT) for the proposed framework with other techniques. The TAT is composed of encryption time, decryption time, KDS time (ECDH mechanism, AES encryption, and decryption time), and Upload and Download time. The tabular values in Fig. 11 indicate that CloudLock takes similar time with the AES and ChaCha20 indicating that it is comparable in overall time consumption with the two. Therefore, our suggested approach presents a compelling proposition by matching the processing speed of current leading-edge methods while consuming less memory and providing heightened security. Furthermore, it introduces the distinct attribute of authentication, marking a novel contribution in this domain.

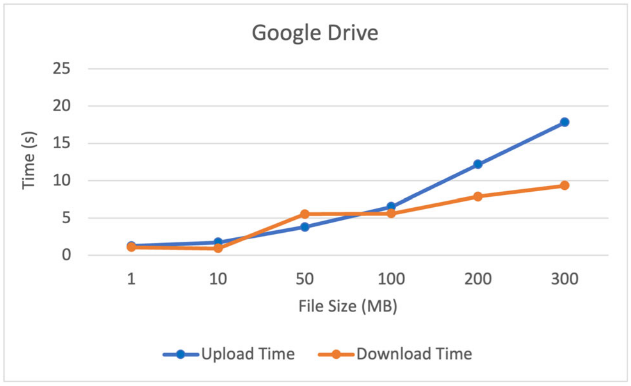


Fig. 9 Upload and download time for Google drive with proposed framework on CSV data

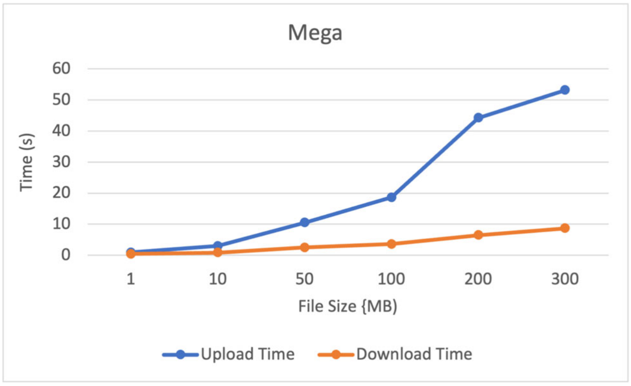


Fig. 10 Upload and download time for Mega with proposed framework on CSV data

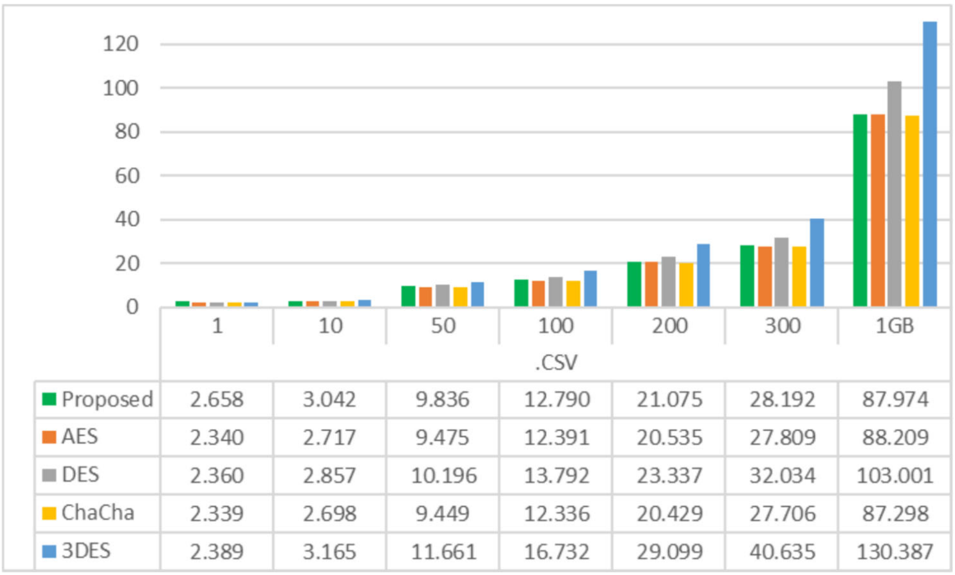
6.3 Security analysis for single-user

In a specific illustration, the security of ChaCha20-Poly1305 for a single user can be inferred from its theorem on multi-user security (Theorem 6.3.1). There is an existing security proof by Procter in the single-user scenario [26]. However, Procter's validation, while in the standard model, assumes different ChaCha20 security measures than our multi-user security display we have endeavored to establish a similar security boundary using identical assumptions in Theorem 6.3.1.

6.3.1 Theorem

Given the parameters n, k, t, c for the ChaCha20-Poly1305 AEAD scheme, consider A as legitimate nonce-abiding adversary conducting a maximum of q_v decryption inquiries. If C^p is the cap on the total size (in t-bit blocks) of each query it makes, then a PRF adversary $A_p rf$ can be found against the ChaCha20 block function CC_block as per the following conditions:

Fig. 11 TAT comparison with Google drive for various schemes



	1	10	50	100	200	300	1GB
🟩 Proposed	2.658	3.042	9.836	12.790	21.075	28.192	87.974
🟧 AES	2.340	2.717	9.475	12.391	20.535	27.809	88.209
▨ DES	2.360	2.857	10.196	13.792	23.337	32.034	103.001
🟨 ChaCha	2.339	2.698	9.449	12.336	20.429	27.706	87.298
🟦 3DES	2.389	3.165	11.661	16.732	29.099	40.635	130.387

$$Adv^{AE}_{ChaCha20-Poly1305}(A) \leq Adv^{PRF}_{CC_b lock}(A_{prf}) + (q_v.c.C^p/2^t \quad (8)$$

In a scenario where A_{prf} submits queries equal to the total block queries made by A, It is important to acknowledge that for minor C^p values, this boundary is precise, as elucidated further in Proposition 6.3.2.

6.3.2 Proposition

Given t as the tag size for Poly1305_Mac and H as its corresponding c-almost Δ - universal hash function, suppose $C^p \leq 5$ is the uppermost quantity of t-bit input blocks in a query for encrypting or verifying through the ChaCha20-Poly1305 AEAD mechanism. We posit the existence of an adversary A that submits a single encryption query and q_v verification queries, such that:

$$Adv^{muAE}_{ChaCha20-Poly1305[\Pi]}(A) \leq (q_v.c.(C^p - 5))/2^{t+4} \quad (9)$$

6.4 Security analysis for multi-user

The forthcoming theorem delineates the multi-user security bounds of ChaCha20-Poly1305 within the ideal permutation model, implying that the ChaCha20 permutation is treated as random. This perspective permits us to encapsulate the computational efforts of the adversary as the number of offline invocations to the ChaCha20 permutation.

6.4.1 Theorem

Consider the AEAD scheme ChaCha20-Poly1305[Π] as illustrated in Algorithm 1, characterized by parameters n, k, t, c, where its fundamental permutations Π are simulated as a random permutation. Suppose A is d-repeating adversary, executing a maximum of p_I ideal permutation queries and q_v validation queries. Also, Let us denote l_m as the upper limit on the size in b-bit blocks (inclusive associated data) that it can probe its encryption and validation oracles. Under these conditions, we can state:

$$\begin{aligned} Adv^{muAE}_{ChaCha20-Poly1305[\Pi]}(A) &\leq (q_v(cl_m + 3))/2^b \\ &+ (d(p_I + q_e))/2^k + (2p_I.(n-k))/2^k \\ &+ (2q_v.(n-k-4b))/2^k + (\sigma_e + q_e)^2/2^{n+1} \\ &+ 1/2^{2b-2} + 1/2^{n-k-2} \end{aligned} \quad (10)$$

In the above, we further require that: n - k

$$\leq 2^{k-2}, \sigma_e \leq (n-k)/6.2^{n-k}, q_v \leq 2^{n-2},$$
$$p_I \leq min((2b-1)/6.(2^{2b}), (n-k-1)/6.2^{n-k}),$$
$$and\, d \leq (2b)/3.2^{2b}$$

7 Conclusion

CloudLock acts as a third-party security provider enabling authentication, downloading, uploading, encryption, decryption, data sharing, data slicing, and data indexing functionalities. It is observed to be a more secure and more reliable technique. Whereas it may take more encryption time than AES and more memory than ChaCha20, the TAT of CloudLock (87.974 s for 1GB file) indicates that it takes a similar time as compared with AES (88.209 s for 1 GB file) and ChaCha20 (87.298 s for 1GB file). Additionally, it uses authentication functionality which is an extension of ChaCha20 so the difference in memory consumption between ChaCha20 (322.302 MB for 1GB TXT file) and CloudLock (367.0938 MB for 1GB TXT file) could be neglected owing to that CloudLock is more invulnerable to attacks than ChaCha20. In all, the proposed framework is considered to be more efficient as it could provide better security in memory-constrained environments. However, onr of the limitation of sir work is the assumption of security aspect of SCSE unit. As no system can be considered completely immune to breaches, especially in the face of evolving threat vectors. Thus in the future, our research aims will concentrate on two primary aspects. First, we aim to bolster the security measures in the SCSE unit further, investigating potential vulnerabilities in the SCSE unit and enhancing its resistance to various forms of cyber-attacks. This will involve reinforcing the unit is capabilities to accurately detect and promptly respond to security threats utilizing AI based anomaly detection, thereby further strengthening its robustness. Secondly, we plan to devise methods for accurately identifying users exhibiting malicious behavior toward the data. This requires developing sophisticated detection mechanisms capable of discerning harmful intent.

Acknowledgements This work was supported in part by Taighde Éireann - Research Ireland under grants 12/RC/2289 P2 (Insight), 21/FFP-A/9174 (SustAIn) and European Union's Horizon Europe research and innovation programme under Grant Agreement No. 101100680 (GN5-1). For the purpose of Open Access, the author has applied a CC BY public copyright licence to any Author Accepted Manuscript version arising from this submission.

Author contributions All authors have contributed equally. All authors have read and approved the final version of the manuscript.

Funding Open Access funding provided by the IReL Consortium. This work was supported in part by Taighde Éireann - Research Ireland under grants 12/RC/2289_P2 (Insight), 21/FFP-A/9174 (SustAIn) and European Union's Horizon Europe research and innovation programme under Grant Agreement No. 101100680 (GN5-1).

Data availability No new data were created or analyzed in this study. Data sharing is not applicable to this article.

Declarations

Conflict of interest No conflict of interest to disclose.

References

1. Grand View Research: Cloud Computing Industry. https://www.grandviewresearch.com/industry-analysis/cloud-computing-industry. Accessed: Mar 15, 2025
2. Hashem, I., Yaqoob, I., Anuar, N.B., Mokhtar, S., Gani, A., Khan, S.U.: The rise of "big data'' on cloud computing: review and open research issues. Inf. Syst. **47**, 98–115 (2015)
3. Verma, P., Tapaswi, S., Godfrey, W.W.: A request aware module using cs-idr to reduce vm level collateral damages caused by ddos attack in cloud environment. Cluster Comput. **24**, 1917–1933 (2021)
4. Almaiah, M.A., Ali, A., Hajjej, F., Pasha, M.F., Alohali, M.A.: A lightweight hybrid deep learning privacy preserving model for fc-based industrial internet of medical things. Sensors **22**(6), 2112 (2022)
5. Almaiah, M.A., Hajjej, F., Ali, A., Pasha, M.F., Almomani, O.: A novel hybrid trustworthy decentralized authentication and data preservation model for digital healthcare IoT based cps. Sensors **22**(4), 1448 (2022)
6. Grand View Research: Multi-Cloud Management Market Report. https://www.grandviewresearch.com/industry-analysis/multi-cloud-management-market-report. Accessed: Mar 15, 2025
7. Seth, D., Nerella, H., Najana, M., Tabbassum, A.: Navigating the multi-cloud maze: benefits, challenges, and future trends. Int. J. Global Innov. Solut. (IJGIS) (2024). https://doi.org/10.21428/e90189c8.8c704fe4
8. AlZain, M.A., Pardede, E., Soh, B., Thom, J.A.: Cloud computing security: from single to multi-clouds. In: 2012 45th Hawaii International Conference on System Sciences, pp. 5490–5499 (2012). IEEE
9. Verma, P., Tapaswi, S., Godfrey, W.W.: An impact analysis and detection of http flooding attack in cloud using bio-inspired clustering approach. Int. J. Swarm Intell. Res. (IJSIR) **12**(1), 29–49 (2021)
10. Bohli, J.-M., Gruschka, N., Jensen, M., Iacono, L.L., Marnau, N.: Security and privacy-enhancing multicloud architectures. IEEE Trans. Dependable Secure Comput. **10**(4), 212–224 (2013)
11. Shaik, K., Narayana Rao, T.V., et al.: Implementation of encryption algorithm for data security in cloud computing. Int. J. Adv. Res. Compute. Sci. **8**(3), 579, (2017)
12. Madanan, M., Patel, P., Agrawal, P., Mudholkar, P., Mudholkar, M., Jaganraja, V.: Security challenges in multi-cloud environments: Solutions and best practices. In: 2024 7th International Conference on Contemporary Computing and Informatics (IC3I), vol. **7**, pp. 1608–1614 (2024). IEEE
13. Razaque, A., Nadimpalli, S.S.V., Vommina, S., Atukuri, D.K., Reddy, D.N., Anne, P., Vegi, D., Malllapu, V.S.: Secure data sharing in multi-clouds. In: 2016 International Conference on Electrical, Electronics, and Optimization Techniques (ICEEOT), pp. 1909–1913 (2016). IEEE
14. Verma, P., Tapaswi, S., Godfrey, W.W.: An adaptive threshold-based attribute selection to classify requests under ddos attack in cloud-based systems. Arab. J. Sci. Eng. **45**, 2813–2834 (2020)
15. Ali, M., Dhamotharan, R., Khan, E., Khan, S.U., Vasilakos, A.V., Li, K., Zomaya, A.Y.: Sedasc: secure data sharing in clouds. IEEE Syst. J. **11**(2), 395–404 (2015)
16. Balasaraswathi, V., Manikandan, S.: Enhanced security for multi-cloud storage using cryptographic data splitting with dynamic approach. In: 2014 IEEE International Conference on Advanced Communications, Control and Computing Technologies, pp. 1190–1194 (2014). IEEE
17. Vaidya, M., Nehe, S.: Data security using data slicing over storage clouds. In: 2015 International Conference on Information Processing (ICIP), pp. 322–325 (2015). IEEE
18. Verma, P., Tapaswi, S., Godfrey, W.W.: Avdr: a framework for migration policy to handle ddos attacked vm in cloud. Wireless Pers. Commun. **115**(2), 1335–1361 (2020)
19. Akinade, A.O., Adepoju, P.A., Ige, A.B., Afolabi, A.I.: Cloud security challenges and solutions: a review of current best practices. Int. J. Multidiscip. Res. Growth Eval. **6**(1), 26–35 (2025)
20. Karnik, N., Kumar, A., Mahajan, P. et al. An efficient technique for securing a multi-cloud storage environment. Int J Syst Assur Eng Manag (2025). https://doi.org/10.1007/s13198-025-02751-2
21. Wang, L.-L., Ke-fei, C., Xian-ping, M., Yong-tao, W.: Efficient and provably-secure certificateless proxy re-encryption scheme for secure cloud data sharing. J. Shanghai Jiaotong Univ. (Chin. Ed.) **19**, 398–405 (2014)
22. Gong, C., Liu, J., Zhang, Q., Chen, H., Gong, Z.: The characteristics of cloud computing. In: 2010 39th International Conference on Parallel Processing Workshops, pp. 275–279 (2010). IEEE
23. Cachin, C., Haas, R., Vukolic, M.: Dependable storage in the intercloud. IBM Res. **3783**, 1–6 (2010)
24. Bessani, A., Correia, M., Quaresma, B., André, F., Sousa, P.: Depsky: dependable and secure storage in a cloud-of-clouds. Acm Trans. Storage (tos) **9**(4), 1–33 (2013)
25. Bhatt, S., Shivarudra, A., Kavuri, S., Mehra, A., Paulraj, B.: Building scalable and secure data ecosystems for multi-cloud architectures. Lett. High Energy Phys. 2024, 11 (2025)
26. Ali, A., Pasha, M.F., Guerrieri, A., Guzzo, A., Sun, X., Saeed, A., Hussain, A., Fortino, G.: A novel homomorphic encryption and consortium blockchain-based hybrid deep learning model for industrial internet of medical things. IEEE Trans. Netw. Sci. Eng. **10**(5), 2402–2418 (2023)
27. Ali, A., Almaiah, M.A., Hajjej, F., Pasha, M.F., Fang, O.H., Khan, R., Teo, J., Zakarya, M.: An industrial IoT-based blockchain-enabled secure searchable encryption approach for healthcare systems using neural network. Sensors **22**(2), 572 (2022)

28. Hajlaoui, N., Bejaoui, C., Ismail, T., Ghanmi, H., Touati, H.: A hybrid architecture for secure data sharing in multi-clouds system. Comput. J. **68**(1), 58–73 (2025)
29. Vaishnav, J., Aulakh, D., Wadhwa, B., Ganesh, D., Singh, J., Sinha, S.K.: Multi-cloud storage augmentation: a novel secured framework for information sharing. Int. J. Syst. Assur. Eng. Manage. 1–10 (2025). https://doi.org/10.1007/s13198-025-02731-6
30. Fabian, B., Ermakova, T., Junghanns, P.: Collaborative and secure sharing of healthcare data in multi-clouds. Inf. Syst. **48**, 132–150 (2015)
31. Viswanath, G., Krishna, P.V.: Hybrid encryption framework for securing big data storage in multi-cloud environment. Evol. Intel. **14**(2), 691–698 (2021)
32. Shamir, A.: How to share a secret. Commun. ACM **22**(11), 612–613 (1979)
33. Blakley, G.R.: Safeguarding cryptographic keys. In: Managing Requirements Knowledge, International Workshop On, pp. 313–313 (1979). IEEE Computer Society
34. Dehkordi, M.H., Mashhadi, S.: An efficient threshold verifiable multi-secret sharing. Comput. Stand. Interfaces **30**(3), 187–190 (2008)
35. Hu, C., Liao, X., Cheng, X.: Verifiable multi-secret sharing based on lfsr sequences. Theoret. Comput. Sci. **445**, 52–62 (2012)
36. Eslami, Z., Ahmadabadi, J.Z.: A verifiable multi-secret sharing scheme based on cellular automata. Inf. Sci. **180**(15), 2889–2894 (2010)
37. Dehkordi, M.H., Mashhadi, S.: New efficient and practical verifiable multi-secret sharing schemes. Inf. Sci. **178**(9), 2262–2274 (2008)
38. Nir, Y., Langley, A.: Request for comments 7539: Chacha20 and poly1305 for ietf protocols. Internet Research Task Force (IRTF). (2015)
39. Procter, G.: A security analysis of the composition of chacha20 and poly1305. Cryptology ePrint Archive (2014)

Nitesh Bharot (Senior Member, IEEE) is the Research Lead and a Senior Postdoctoral Researcher at the Insight SFI Research Centre for Data Analytics, University of Galway. He earned his Ph.D. in Cloud Security from Rabindranath Tagore University (RNTU), India (an NIRF-ranked University). Dr Bharot has secured numerous grants from prominent funding agencies, including the EU, GEANT, SFI, Insight, MHRD, and MPCST. Additionally, he has contributed as a Technical Program Committee Member, Speaker, and Reviewer for various IEEE conferences and journals. His research interests encompass Cyber Security, AI/ML, Healthcare, and Industry.

Nakul Mehta is a Research Ireland funded Ph.D. Researcher with the Centre's for Research Training in Artificial Intelligence. He has completed Bachelors in Information Technology from DR BR Ambedkar National Institute of Technology Jalandhar (NITJ). Nakul, has experience as a Data Scientist at Aramex and also completed a research internship at University of Galway, Ireland in the past.

John Breslin (M'94–SM'16) is a Professor in Electronic Engineering at the University of Galway, where he is Director of the TechInnovate and AgInnovate entrepreneurship programs. Associated with two SFI Research Centres, he is a Co-Principal Investigator at Insight (Data Analytics) and a Funded Investigator at VistaMilk (AgTech). He received a Bachelor of Electronic Engineering in 1994 and a Ph.D. in Electronic Engineering in 2002, both from the University of Galway. He has co-authored around 300 publications, including the books "The Social Semantic Web'', "Social Semantic Web Mining'', and the "Old Ireland in Colour'' trilogy. He co-created the SIOC framework, implemented in hundreds of applications (by Yahoo, Boeing, Vodafone, etc.) on at least 65,000 websites with 35 million data instances. He is co-founder of the PorterShed, boards.ie and adverts.ie. John's homepage: https://johnbreslin.com/.

Priyanka Verma (Senior Member, IEEE) received her Ph.D. from the Atal Bihari Vajpayee Indian Institute of Information Technology and Management, India. She completed the Leadership and Innovation Program at the Massachusetts Institute of Technology, USA. She was formerly an Assistant Professor at the Maulana Azad National Institute of Technology Bhopal. She held a Marie Skłodowska-Curie Action Fellowship. Currently, she works as a Assistant Professor at School of Computer Science, University of Galway, Ireland. She has also been a Visiting Research Scholar at Anglia Ruskin University, Chelmsford, U.K. She has co-authored numerous publications in leading journals and conferences. Her research interests include cybersecurity, IIoT, smart manufacturing, AI/ML, cloud, and edge computing. She has received many awards at both national and international levels. She is a conference speaker and has delivered expert lectures in many countries and reviewer of many reputed Journal.