Journal of Science Engineering Technology and Management Science
Volume 02, Issue 08, August 2025 *ISSN: 3049-0952*
www.jsetms.com *DOI: 10.64771/jsetms.2025.v02.i08.613-624*

ENHANCING CLOUD DATA SECURITY AND PREVENTING IMPERSONATION ATTACKS

N. Arpitha[1],S. Deepika[2], K. Sujith[2], Ravula Hari Krishna[2], Korra Sai Kiran[2]
[1]Assistant Professor, [2]UG Student, [1,2] Department of Computer Science and Engineering
[1,2] Sree Dattha Institute of Engineering and Science, Sheriguda, Ibrahimpatnam, 501510, Telangana.

To Cite this Article
N. Arpitha, S. Deepika, K. Sujith, Ravula Hari Krishna, Korra Sai Kiran, "Enhancing Cloud Data Security And Preventing Impersonation Attacks", Journal of Science Engineering Technology and Management Science, Vol. 02, Issue 08, August 2025,pp: 613-624, DOI: http://doi.org/10.64771/jsetms.2025.v02.i08.pp613-624
Submitted: 15-07-2025 *Accepted: 21-08-2025* *Published: 28-08-2025*

ABSTRACT
In India, data breaches and unauthorized access to sensitive files have surged, with over 700 million digital records compromised in the last decade. Manual file-sharing methods remain prevalent in sectors like healthcare and legal services, where confidentiality is paramount. The objective is to develop a secure, biometric-enabled file-sharing system using ECIES and ChaCha20 encryption to ensure user authentication, protect data confidentiality, and evaluate encryption performance in real-time. In traditional manual file-sharing systems, sensitive documents are exchanged via physical media (e.g., USB drives) or unsecured email. Access is typically controlled through basic passwords or by physical custody. There is no integrated encryption mechanism, and files remain vulnerable to unauthorized access, theft, or loss. Authentication relies on easily compromised credentials, and there is no performance benchmarking for applied cryptographic methods. Manual systems suffer from weak authentication, relying solely on passwords that can be guessed or stolen. File transmission is unsecured, and once data is leaked, it's irretrievable. Access control is poorly enforced, and no encryption ensures content confidentiality. Furthermore, there's no mechanism to track or benchmark encryption performance, limiting scalability and trust in sensitive environments. The research aims to overcome limitations of manual systems by introducing multi-factor authentication using fingerprint hashing and integrating real-time encryption performance evaluation The proposed system integrates ECIES and ChaCha20 to provide a secure, performance-optimized file-sharing platform. Upon registration, users submit SHA-256–hashed usernames and fingerprint images to enable biometric login. Uploaded files are encrypted using ECIES with the user's public key to ensure only authorized decryption. Simultaneously, ChaCha20 encrypts the file to benchmark speed, revealing that it outperforms ECIES for large files. All metadata—owner, filename, and access type—is stored in a MySQL database, and access is controlled per file. Real-time graphing with Matplotlib compares encryption speeds, allowing administrators to make informed choices. This hybrid approach ensures strong security and operational efficiency.

Keywords: Cloud data security, privacy preservation, encryption, impersonation prevention, robust authentication, biometric security, ECIES, ChaCha20, secure file sharing, anonymous access, cryptographic framework.

1. INTRODUCTION

Cyber–Physical–Social Systems (CPSSs) are integrated environments that combine data and functionality across cyber, physical, and social domains, leveraging sensors, actuators, computational resources, and human interaction to support intelligent decision-making and real-world applications such as smart cities, intelligent transportation, and healthcare. Within this context, the proposed research addresses the limitations of traditional file-sharing systems, which typically rely on password-based authentication and centralized key management, exposing them to significant risks like credential theft and compromised data confidentiality. The motivation behind this work stems from the growing need for stronger security mechanisms—including multi-factor authentication using SHA-256–hashed fingerprint images—and empirical benchmarking of encryption schemes. The research aims to develop a Django-based, open-source platform that integrates ECC/ECIES for robust file encryption, implements biometric verification for enhanced authentication, and uses ChaCha20 encryption in parallel to evaluate computational efficiency through real-time performance graphs generated with Matplotlib. The system also supports access control through a MySQL database, ensuring that private files remain restricted to their owners, while public files are freely accessible. The significance of the platform lies in its compliance with data protection regulations (e.g., HIPAA, GDPR), its resistance to current cryptographic attacks, and its cost-effective implementation using freely available libraries. Practical applications span healthcare, legal, corporate, defense, and academic sectors, where secure file sharing, authenticated access, and transparent encryption benchmarking are essential for data integrity, privacy, and informed system optimization.

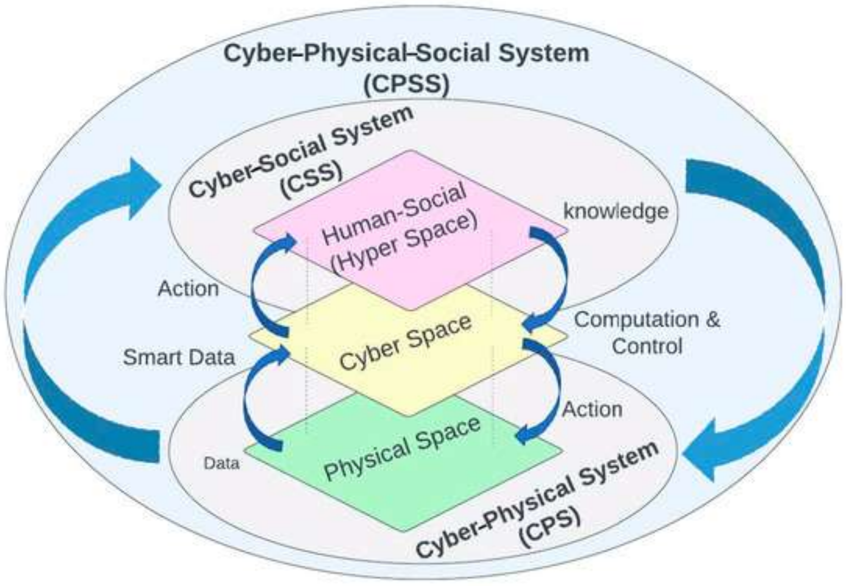

Cyber-Physical-Social System (CPSS)
Cyber-Social System (CSS)
Human-Social (Hyper Space)
knowledge
Action
Computation & Control
Smart Data
Cyber Space
Action
Data
Physical Space
Cyber-Physical System (CPS)

Fig. 1: A Cyber–Physical–Social System (CPSS)

2. LITERATURE SURVEY

In In recent years, the hasty evolution of the Internet-of-Things (IoT) is specified as cyber-physical systems (CPSs), has increased the digital innovation and improved the environment of human living. Cyber-physical-social systems (CPSSs) encompassing the physical, social, and cyber world, are escalating in every aspect of our daily lives. The provision of an efficient living environment by offering high quality personalized and proactive services to humans is one of the primary purposes of CPSSs. The massive amount of data in our daily lives is constantly produced from cyber, physical, and social worlds. Data is the essential pivoting point of our research, which is flowing around these three worlds and maintaining patterns of all our daily life aspects. However, vast amounts of data acquired from CPSSs are usually complex, low-quality, noisy, and redundant, which causes unexceptional challenges for offering CPSSs services. The comprehensive analysis of cloud computing based big data is essential. Moreover, secure communication for a cloud computing environment is also required to offer high quality and reliable services.

A new type of storage, referred to as cloud storage, has emerged with the evolution of the cloud computing paradigm. The adoption of cloud storage, particularly leveraging on the services of public or private cloud data storage, includes not only several benefits regarding reliability, flexibility, and scalability but also infers new challenges that are related to data privacy, protection, and security. Practically, public clouds provide several advantages such as no chain of risks towards infrastructure providers and no cost of the initial investment. However, efficient control on network, data, and security settings is lacked by the public that diminishes the interest of acquiring the services of the cloud. They also increase the challenges of data trust, security, and privacy. On the other hand, the private cloud reflects the quality of the service factors that includes reliability, security, and performance. Moreover, private cloud offers an opportunity to many organizations for the implementation of their own security acquiescence without depending on security measures of cloud providers. Whenever any sensitive data is involved, the services of a private cloud storage are used.

For sharing the data, the DO should take part in that process, to manage the convenience of users and only authorized access should be allowed in the model of cloud storage. However, the methods of data sharing (i.e., proxy encryption [5], [6] and Attribute Base Encryption (ABE) [7], [8]), which are used in the public cloud storages, are not convenient for use in storage model of private cloud. The reason is that the DO is not aware of the specific user's identity user who needs to acquire data. The DO also provides a secure mechanism of group data-sharing.

The numerous collections of technologies configure cloud storage, and the cloud server manages it. So, the security of outsourced data relies on the flexible and secure data services in cloud storage that facilitates data security, privacy, and authentication to a user for different operations of data. The data distribution, data storage, and data access are such components of data service which are required for cloud storage to give secure solutions such as auditing, integrity, and flexible sharing of data [11], [12]. The existing solutions for cloud storage are concentrated on offering an efficient access control method to outsourced data by introducing a key management system. For flexible data access, this system enforces a self-reliant authorization with legitimate users [13], [14]. The job of data owner is different from the providers of cloud storage due to the outsourcing of data service management, and the users do not interact with the data owner for offering authorized data access.

3. PROPOSED METHODOLOGY

This project presents a Django-based secure file-sharing application integrated with a MySQL backend and advanced cryptographic techniques, aiming to ensure encrypted file transmission and benchmark the performance of Elliptic Curve Cryptography (ECC) and ChaCha20 algorithms. Users register by submitting personal details along with a fingerprint image, where both the username and fingerprint are hashed using SHA-256, and credentials are stored in a database after duplication checks. Upon successful login, verified by matching stored and input hashes and plaintext password, users can upload files marked as either "Public" or "Private." During upload, the system encrypts the file using ECC (storing encryption time) and separately performs ChaCha20 encryption for performance comparison, displaying results via a Matplotlib-generated bar chart. Encrypted files are saved to disk, and metadata is recorded in the database. Public files can be downloaded by any user, while private files are restricted to their respective owners. Downloads involve ECC decryption in-memory before serving the original file to the user. The system also allows users to view a permissions-based download table, change passwords through a simple form (though with a basic SQL update query), and visualize encryption performance metrics. ECC key management ensures keys are generated or retrieved when needed, and all encrypted data operations are efficiently handled through dedicated encryption/decryption wrappers. This comprehensive approach ensures secure storage, user-level access control, and cryptographic performance transparency within a user-friendly web interface.

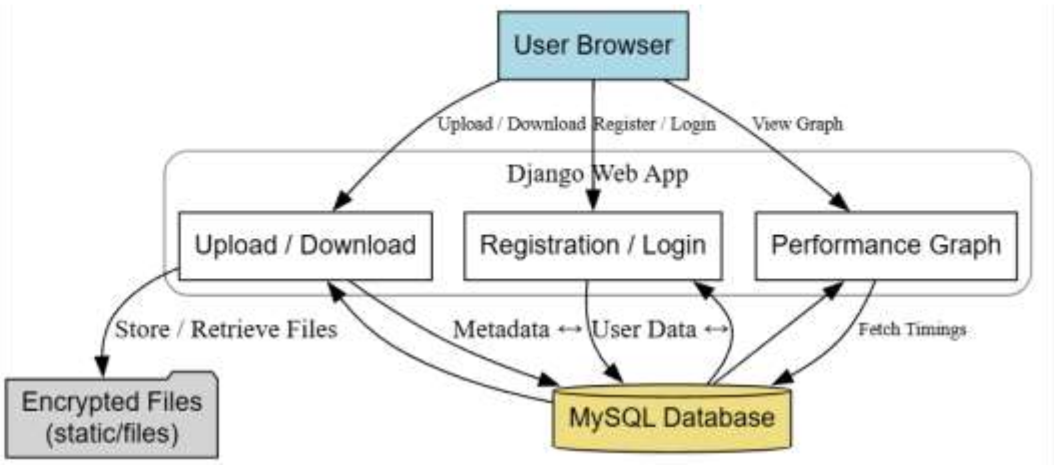


Fig. 2: Proposed system architecture of enhanced data privacy and security in cloud storage.

Proposed workflow

The secure file-sharing application begins with user registration, where navigating to /Register.html presents a form requesting a username, password, contact, email, address, and a fingerprint image. Upon form submission to /Signup, the server hashes the username and fingerprint using SHA-256, checks for existing entries in the newuser table, and, if none exist, stores the new user details, displaying a “Signup Process Completed” message. For login, users access /Login.html, input their credentials and fingerprint, and submit the form to /UserLogin, where the server verifies the SHA-256 hashes of the username and fingerprint and matches the plaintext password to authenticate, redirecting successful users to UserScreen.html with a personalized greeting. From there, users may upload files via /UploadFile.html, choosing a file and access type (“Public” or “Private”), which are submitted to /UploadFileAction. The system encrypts the file using ECC (recording the time), then benchmarks ChaCha20 encryption, storing relevant metadata in the share table and saving the ECC-encrypted file under static/files/. Upon success, users are redirected back with a confirmation message. File access is managed through /DownloadFile, which fetches and displays all shared files in an HTML table, allowing downloads only if permission criteria are met. Downloading a file involves a request to /DownloadAction?name=<filename>, where the ECC-encrypted content is read, decrypted in memory, and returned as a browser download. To analyze encryption performance, users can access /Graph, where a Matplotlib bar chart compares ECC and ChaCha20 encryption durations from the latest upload, rendered in ViewGraph.html as a base64-encoded image. Additionally, users may navigate to /ChangePassword.html to update their password by submitting old and new credentials to /ChangePasswordAction, where a basic SQL update checks for a match and modifies the record if validated, showing a corresponding success or failure message.

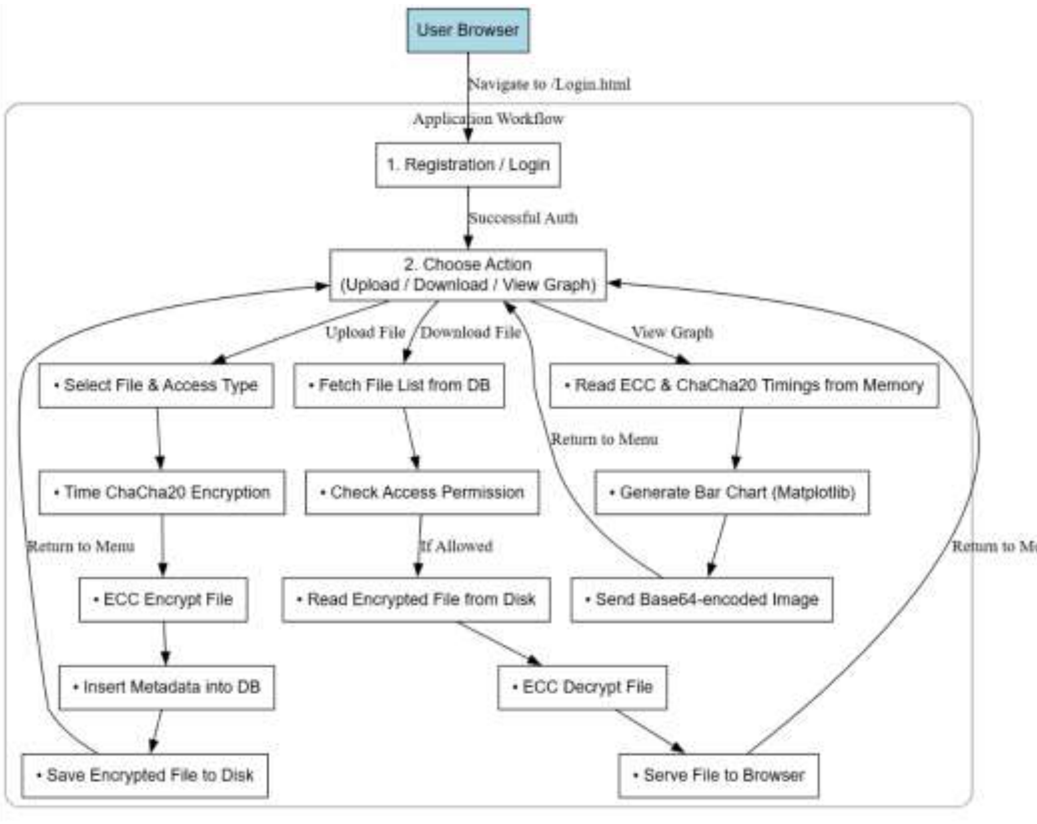

User Browser
Navigate to /Login.html
Application Workflow
1. Registration / Login
Successful Auth
2. Choose Action (Upload / Download / View Graph)
Upload File
Download File
View Graph
• Select File & Access Type
• Fetch File List from DB
• Read ECC & ChaCha20 Timings from Memory
Return to Menu
• Time ChaCha20 Encryption
• Check Access Permission
• Generate Bar Chart (Matplotlib)
Return to Menu
If Allowed
Return to Menu
• ECC Encrypt File
• Read Encrypted File from Disk
• Send Base64-encoded Image
• Insert Metadata into DB
• ECC Decrypt File
• Save Encrypted File to Disk
• Serve File to Browser

Fig. 3: Proposed workflow of enhanced data privacy and security in cloud storage.

Cryptography Models/ Elliptic Curve Cryptography (ECC / ECIES)

Elliptic Curve Cryptography (ECC) is a public-key system whose security derives from the intractability of the Elliptic Curve Discrete Logarithm Problem (ECDLP) over finite fields, and ECIES (Elliptic Curve Integrated Encryption Scheme) leverages ECC to create a hybrid encryption protocol that securely handles arbitrary-length data: during key generation, a random private scalar d is chosen in the interval [1,n−1] (where n is the curve's order), the corresponding public point Q = d·G (is computed using the generator G), and both keys are serialized in hex to pvt.key and pri.key; in encryption, the sender retrieves the recipient's public key Q, generates an ephemeral key pair (r and R = r·G), computes the shared secret S = r·Q, derives symmetric encryption and MAC keys from S via a KDF, encrypts the plaintext with a symmetric cipher (e.g., AES-GCM or ChaCha20-Poly1305), computes an authentication tag, and outputs the tuple {R, ciphertext, MAC}; decryption mirrors these steps by parsing R, ciphertext, and MAC from the package, computing S = d·R using the recipient's private key d, deriving the same symmetric keys, verifying the MAC to ensure integrity, and finally decrypting the ciphertext to recover the original plaintext.

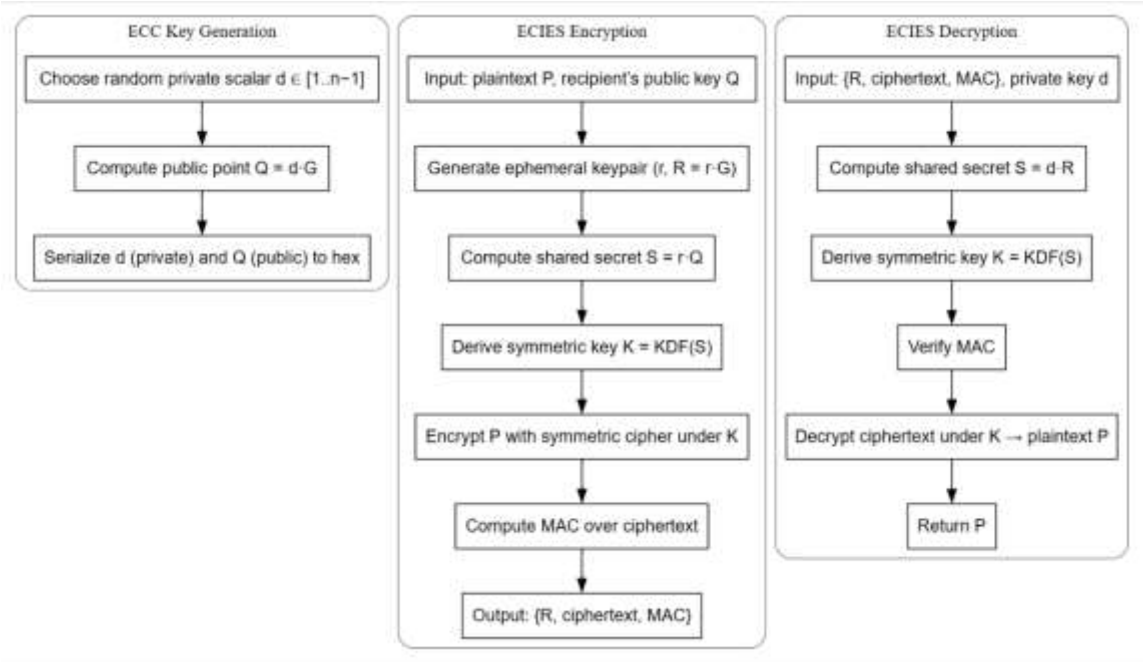

ECC Key Generation
Choose random private scalar d ∈ [1..n−1]
Compute public point Q = d·G
Serialize d (private) and Q (public) to hex
ECIES Encryption
Input: plaintext P, recipient's public key Q
Generate ephemeral keypair (r, R = r·G)
Compute shared secret S = r·Q
Derive symmetric key K = KDF(S)
Encrypt P with symmetric cipher under K
Compute MAC over ciphertext
Output: {R, ciphertext, MAC}
ECIES Decryption
Input: {R, ciphertext, MAC}, private key d
Compute shared secret S = d·R
Derive symmetric key K = KDF(S)
Verify MAC
Decrypt ciphertext under K → plaintext P
Return P

Fig. 4: Internal operational flow of ECC/ECIES.

ChaCha20 Symmetric Stream Cipher

ChaCha20 is a high-speed, secure stream cipher developed by Daniel J. Bernstein, designed to efficiently encrypt data using simple yet robust operations. It operates on 512-bit blocks and generates a pseudorandom keystream that is XORed with plaintext to produce ciphertext. The cipher uses a 256-bit (32-byte) secret key, a 96-bit (12-byte) nonce, and a 32-bit block counter. The encryption process begins with initializing a 16-word (512-bit) internal state composed of a fixed constant (e.g., "expand 32-byte k"), the 256-bit key, the block counter, and the nonce. This state undergoes 20 rounds of transformation through the ChaCha quarter-round function, which involves modular addition, XOR, and bitwise rotation operations for strong diffusion and confusion. After processing, the final state is added to the original state (word-wise modulo 2^{32}), and the result is serialized into little-endian format to produce a 64-byte keystream block. This design ensures high performance, strong cryptographic security, and resistance to side-channel attacks, making ChaCha20 well-suited for modern secure communications.

Encryption / Decryption:

ChaCha20, being a symmetric stream cipher, performs encryption and decryption using the same operation—XORing the data with a pseudorandom keystream—so that ciphertext = plaintext $\oplus$ keystream and plaintext = ciphertext $\oplus$ keystream. In this project, file encryption using ChaCha20 involves generating a random 256-bit (32-byte) key via get_random_bytes(32) and initializing the cipher with ChaCha20.new(key=key), where the PyCryptodome library automatically selects a secure 64-bit or 96-bit nonce. The file data is then encrypted by calling cipher.encrypt(filedata), which produces the ciphertext. However, this implementation is used solely for benchmarking purposes; the ChaCha20 key and resulting ciphertext are not stored—instead, the system measures and records only the time taken for the encryption process, saving this duration in the propose variable for performance comparison.

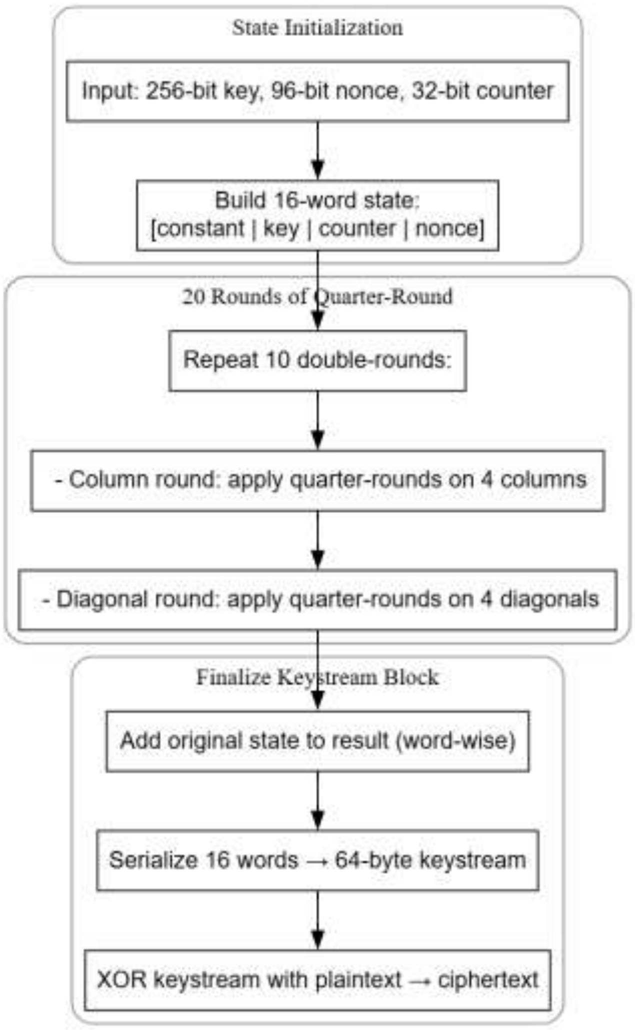

State Initialization
Input: 256-bit key, 96-bit nonce, 32-bit counter
Build 16-word state:
[constant | key | counter | nonce]
20 Rounds of Quarter-Round
Repeat 10 double-rounds:
- Column round: apply quarter-rounds on 4 columns
- Diagonal round: apply quarter-rounds on 4 diagonals
Finalize Keystream Block
Add original state to result (word-wise)
Serialize 16 words → 64-byte keystream
XOR keystream with plaintext → ciphertext

Fig. 5: Internal operational flow of ChaCha20.

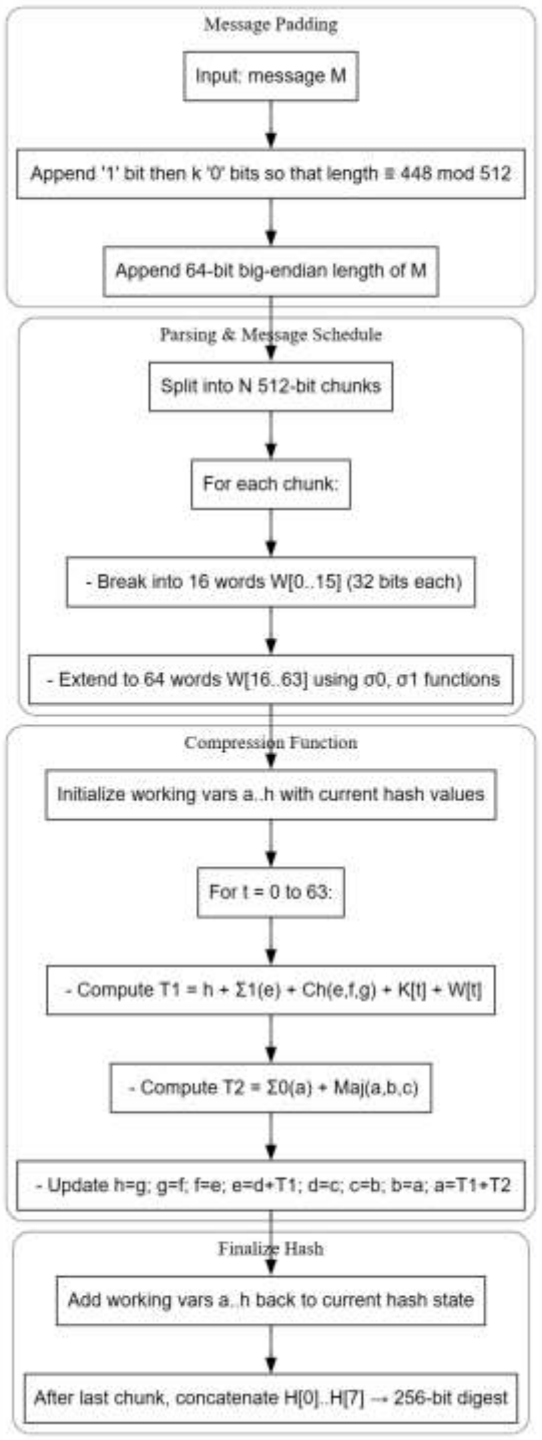

Message Padding
Input: message M
Append '1' bit then k '0' bits so that length ≡ 448 mod 512
Append 64-bit big-endian length of M
Parsing & Message Schedule
Split into N 512-bit chunks
For each chunk:
- Break into 16 words W[0..15] (32 bits each)
- Extend to 64 words W[16..63] using σ0, σ1 functions
Compression Function
Initialize working vars a..h with current hash values
For t = 0 to 63:
- Compute T1 = h + Σ1(e) + Ch(e,f,g) + K[t] + W[t]
- Compute T2 = Σ0(a) + Maj(a,b,c)
- Update h=g; g=f; f=e; e=d+T1; d=c; c=b; b=a; a=T1+T2
Finalize Hash
Add working vars a..h back to current hash state
After last chunk, concatenate H[0]..H[7] → 256-bit digest

Fig. 6: SHA-256 internal operation.

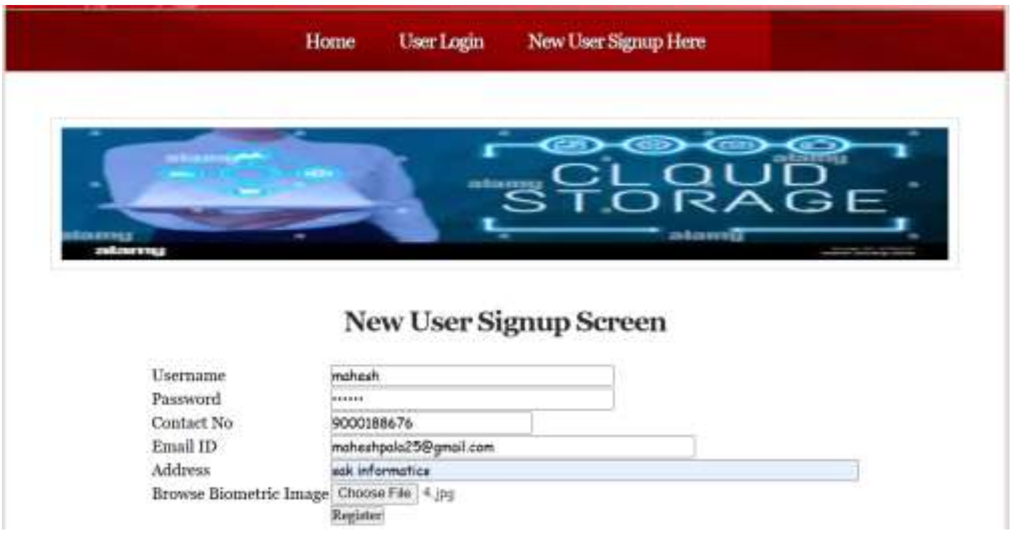
SHA-256 Hash Function

SHA-256 is a one-way cryptographic hash function from the SHA-2 family. It maps an arbitrary-length input to a fixed 256-bit (32-byte) digest. It is collision-resistant (finding two messages with the same hash is computationally infeasible) and preimage-resistant (given a hash, finding any message that hashes to it is infeasible). The main purpose of this algorithm is as follows:

- By hashing the username, the system never retains a raw, guessable username in the database; it only holds a SHA-256 digest.
- By hashing fingerprint images, the actual images aren't stored—only their fixed-length digests—so the database doesn't contain raw biometric data.

- During login, the exact match of the freshly computed hash against the stored hash serves as authentication (though storing plaintext passwords is still a security risk).

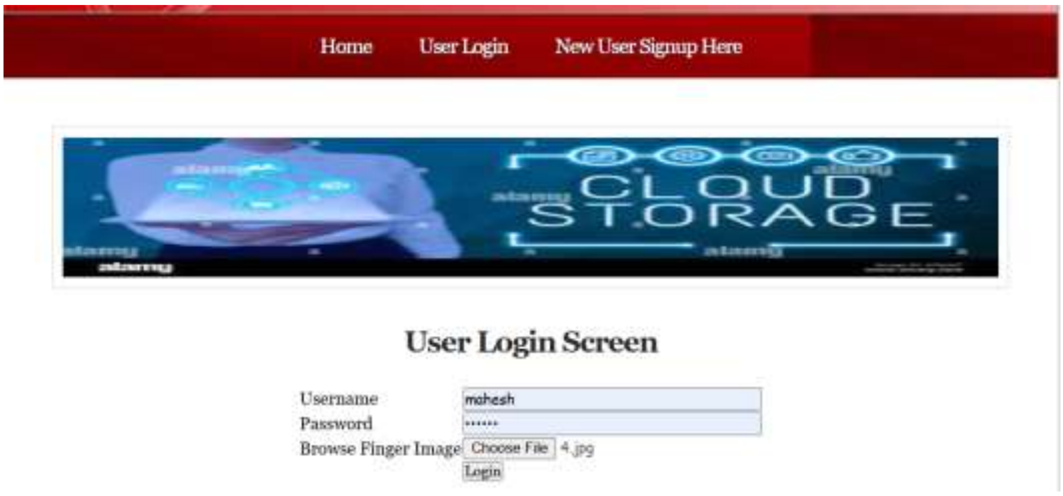
4. RESULTS



New User Signup Screen

Username	mahesh
Password	••••••
Contact No	9000188676
Email ID	maheshpala25@gmail.com
Address	ssk informatics
Browse Biometric Image	Choose File 4.jpg
	Register

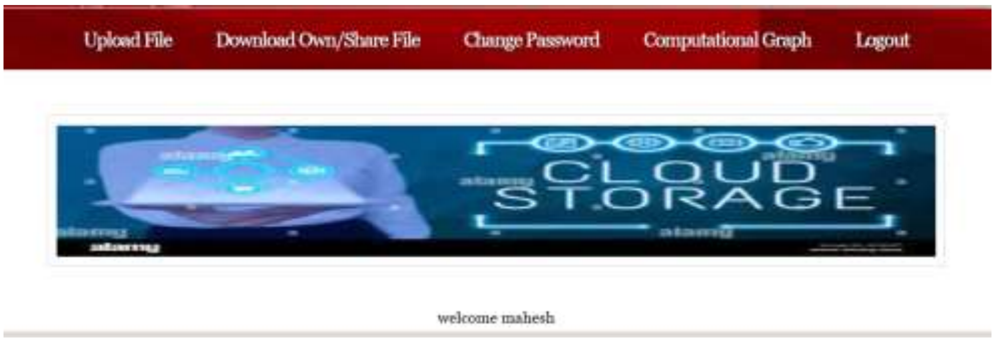
Fig. 7: New user sign up page of proposed method contains the username, password, contact, email ID, address, and choose biometric image.



User Login Screen

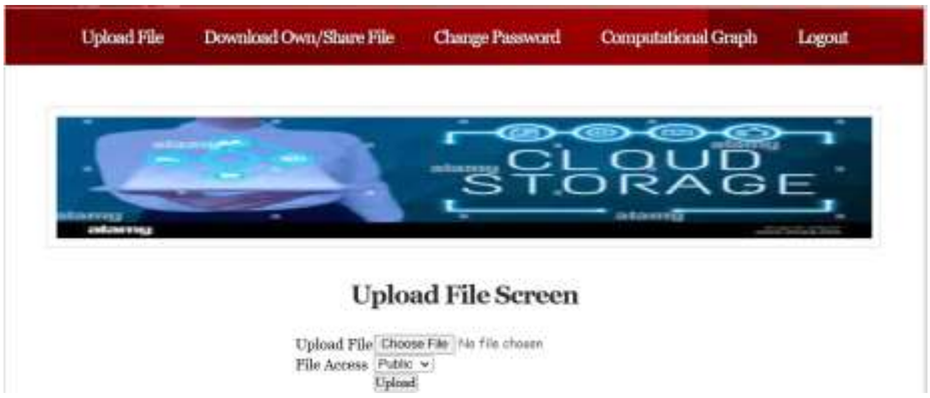
Username	mahesh
Password	••••••
Browse Finger Image	Choose File 4.jpg
	Login

Fig. 8: Use login page



welcome mahesh

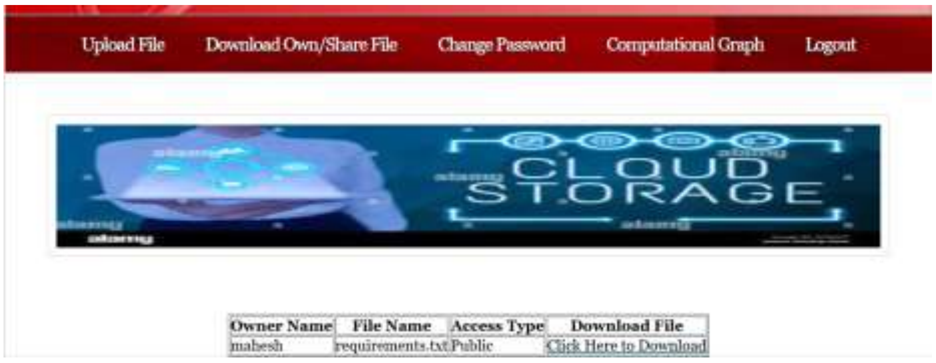
Fig. 9: Web application showing the user functionalities after login.



Upload File: No file chosen
File Access: Public

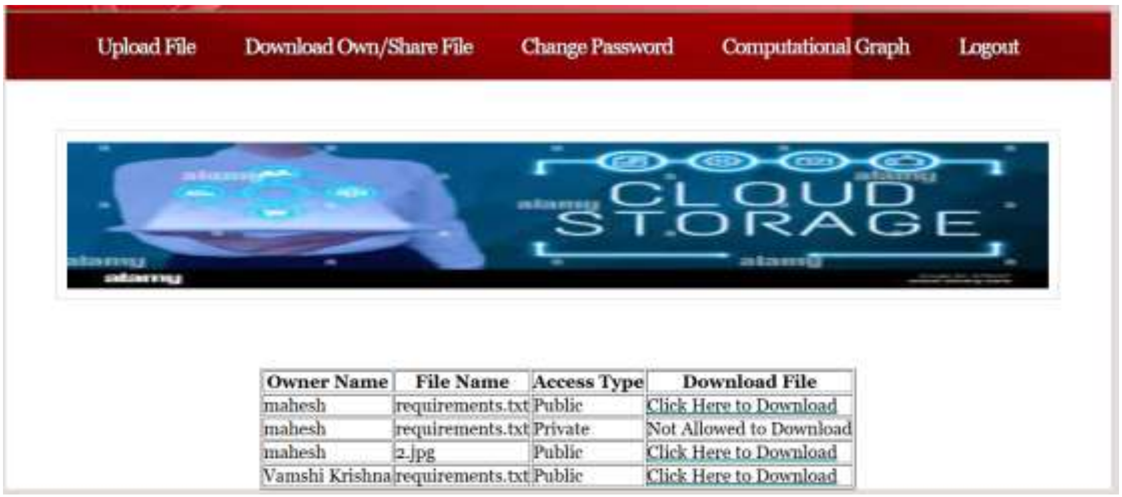
Fig. 10: File upload operation showing the upload file and file access selection (private or public)

The upload screen (Fig. 10) presents a user-friendly, centered form designed for secure file submission, featuring three main controls: a “Select File” button that opens the local system dialog for choosing any file, with the selected filename displayed next to it; an “Access Type” field provided as a dropdown or radio-button group offering two options—“Public,” allowing downloads by any user, and “Private,” restricting access to the uploader only; and a prominently styled “Upload” button positioned at the bottom of the form. Once a file is chosen and the access level is selected, clicking “Upload” initiates the ECC encryption process, stores the file metadata, and may briefly show a status bar or spinner to indicate progress. Upon successful encryption and storage, a green success message—such as “File successfully loaded to cloud”—is displayed at the top of the screen, confirming completion to the user.



Owner Name	File Name	Access Type	Download File
mahesh	requirements.txt	Public	Click Here to Download

(a)



Owner Name	File Name	Access Type	Download File
mahesh	requirements.txt	Public	Click Here to Download
mahesh	requirements.txt	Private	Not Allowed to Download
mahesh	2.jpg	Public	Click Here to Download
Vamshi Krishna	requirements.txt	Public	Click Here to Download

(b)

Fig. 11: Download own/shared file. (a) public type. (b) both public and private type.

Figure 12 is a simple, embedded PNG bar graph that visually compares encryption performance between two cryptographic algorithms. The X-axis displays two labeled categories: “ECC Computation” and “ChaCha Computation,” representing the encryption techniques used. The Y-axis measures computation time in seconds, marked with evenly spaced tick values such as 0.0, 0.5, 1.0, and so on, providing a clear scale for performance evaluation. The graph contains two vertical bars: the first bar, typically colored blue or rendered in Matplotlib’s default color scheme, indicates the time taken to encrypt the most recently uploaded file using ECC; the second bar, styled similarly, shows the time required by ChaCha20 for the same file. The height of each bar directly corresponds to the recorded encryption duration, offering users and administrators a quick visual insight into the relative computational efficiency of the two algorithms.

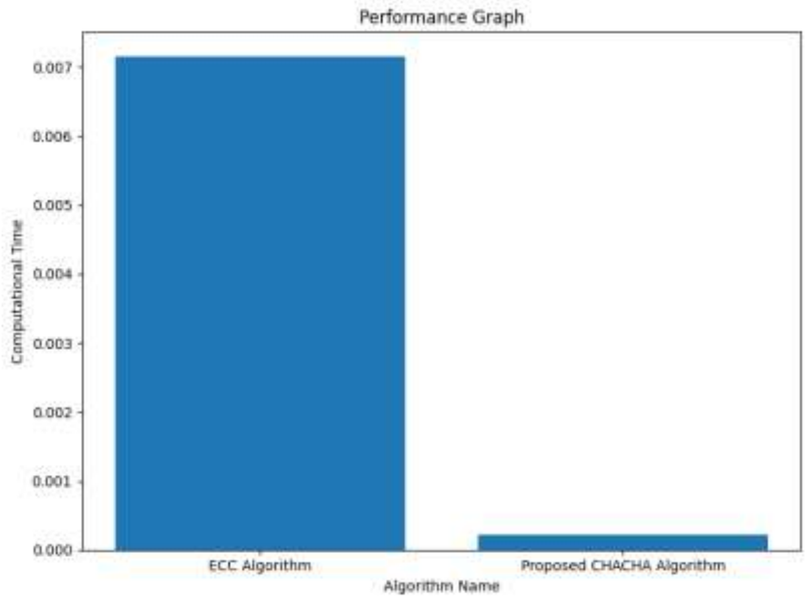


Fig. 12: Performance of computation of time obtained using ECC, and ChaCha algorithms.

5. CONCLUSION

The proposed biometric-enabled, lightweight secure file-sharing system successfully integrates multiple layers of security and performance transparency. By requiring users to register with a SHA-256–hashed username, a plaintext password, and a fingerprint image (also hashed), the platform prevents unauthorized access even if passwords are compromised. Elliptic Curve Integrated Encryption Scheme (ECIES) ensures that every uploaded file is encrypted under the user’s public key, making data at rest unintelligible without the corresponding private key. Storing encrypted files in a controlled static directory, combined with per-file “public” or “private” access flags in the MySQL database, enforces strict download permissions. Performance benchmarking with ChaCha20 encryption demonstrates the relative computational overhead of public-key encryption versus a modern, high-speed symmetric cipher. The Matplotlib-generated bar chart—embedded as a Base64 PNG—gives administrators real-time insight into encryption times, empowering data-driven decisions about algorithmic optimization as file volumes grow. Overall, the system balances robust security and usability: users enjoy a straightforward web interface (registration, login, upload, download, graph view), while administrators gain visibility into cryptographic performance. Direct use of open-source libraries (Django, PyMySQL, ecies, PyCryptodome) and standard hashing (SHA-256) reduces development cost and eases deployment on commodity Linux servers. The strict separation of private ECC key storage, encrypted file storage, and database metadata—combined with HTTPS/TLS transport—ensures confidentiality, integrity, and authenticity across multiple threat vectors. In sum, this project demonstrates a practical, scalable approach to secure, biometric-backed file sharing with built-in performance insights, making it suitable for environments that require high assurance (e.g., healthcare, finance, legal) without prohibitive overhead.

REFERENCES

[1] Zhu, Y.; Tan, Y.; Li, R.; Luo, X. Cyber-physical-social-thinking modeling and computing for geological information service system. In Proceedings of the International Conference on Identification, Information, and Knowledge in the Internet of Things (IIKI), Beijing, China, 22–23 October 2015.

[2] Xiong, G.; Zhu, F.; Liu, X.; Dong, X.; Huang, W.; Chen, S.; Zhao, K. Cyber-physical-social system in intelligent transportation. *IEEE CAA J. Autom. Sin.* 2015, *2*, 320–333.

[3] Cassandras, C.G. Smart cities as cyber-physical social systems. *Engineering* 2016, *2*, 156–158.

[4] Gharib, M.; Lollini, P.; Bondavalli, A. Towards an Approach for Analyzing Trust in Cyber-Physical-Social Systems. In Proceedings of the 12th System of Systems Engineering Conference (SoSE), Waikoloa, HI, USA, 18–21 June 2017; pp. 18–21.

[5] B. Libert and D. Vergnaud, Unidirectional chosen-ciphertext secure proxy re-encryption, IEEE Trans. Inf. Theory, vol. 57, no. 3, pp. 17861802, Mar. 2011.

[6] J. Katz and M. Yung, Applied Cryptography and Network Security: 5th International Conference, ACNS 2007, Zhuhai, China, June 5-8, 2007, Proceedings, vol. 4521. Berlin, Germany: Springer, 2007.

[7] J. Hur and D. K. Noh, Attribute-based access control with ef cient revocation in data outsourcing systems, IEEE Trans. Parallel Distrib. Syst., vol. 22, no. 7, pp. 12141221, Jul. 2011.

[8] W. Li, K. Xue, Y. Xue, and J. Hong, TMACS: A robust and veri able threshold multi-authority access control system in public cloud stor age, IEEE Trans. Parallel Distrib. Syst., vol. 27, no. 5, pp. 14841496, May 2016.

[9] C. Wang, Z.-G. Qin, J. Peng, and J. Wang, A novel encryption scheme for data deduplication system, in Proc. Int. Conf. Commun., Circuits Syst. (ICCCAS), Jul. 2010, pp. 265269.

[10] D. Tiwari, G. K. Chaturvedi, and G. R. Gangadharan, ACDAS: Authen ticated controlled data access and sharing scheme for cloud storage, Int. J. Commun. Syst., vol. 32, no. 15, p. e4072, Aug. 2019.

[11] Z. Fu, K. Ren, J. Shu, X. Sun, and F. Huang, Enabling personalized search over encrypted outsourced data with efficiency improvement, IEEE Trans. Parallel Distrib. Syst., vol. 27, no. 9, pp. 25462559, Sep. 2016.

[12] M. Sookhak, Dynamic remote data auditing for securing big data storage in cloud computing, Ph.D. dissertation, Univ. Malaya, Kuala Lumpur, Malaysia, 2015.

[13] D. Thilakanathan, S. Chen, S. Nepal, and R. A. Calvo, Secure data sharing in the cloud, in Security, Privacy and Trust in Cloud Systems. Berlin, Germany: Springer, 2014, pp. 4572.

[14] J. Li, J. Li, Z. Liu, and C. Jia, Enabling efficient and secure data sharing in cloud computing, Concurrency Comput., Pract. Exper., vol. 26, no. 5, pp. 10521066, 2014.