An Improved Authentication Scheme for Remote Data Access and Sharing Over Cloud Storage in Cyber-Physical-Social-Systems

Mrs. CH Aradhana[1], Mohammed Abdul Aziz[2], Abdullah Shareef[3], Ahmed Maqsood Ali[4]
[1]*Assistant Professor, Dept. of CSE-AIML, Lords Institute of Engineering and Technology*
[2,3,4]*B.E. Students, Dept. of CSE-AIML, Lords Institute of Engineering and Technology*

***Abstract*—Cyber-Physical-Social Systems (CPSSs) encompass the physical, social, and cyber world, generating massive amounts of sensitive data that must be stored and accessed through cloud infrastructure. Existing cloud-storage authentication schemes expose user identities and fail to resist well-known attacks such as replay, man-in-the-middle, and impersonation. This paper proposes an improved authentication scheme that anonymizes user identities via SHA-256 hashing, incorporates biometric verification, and encrypts data using Elliptic Curve Cryptography (ECC) for the base system and ChaCha20 as a high- performance extension. The system enables secure file upload, controlled sharing (public/private), and authenticated download, while a computation-time graph provides empirical comparison between ECC and ChaCha20. Experimental results confirm that ChaCha20 achieves lower computation time than ECC while maintaining equivalent security strength, making it well-suited for resource-constrained IoT-CPSS environments. The proposed scheme protects against internal cloud-server threats, preserves user anonymity, and scales effectively to multiple data owners and mobile users.**

***Index Terms*—Cloud Storage, Authentication, Cyber-Physical-Social Systems, ECC, ChaCha20, Biometric Verification, Data Privacy, Federated Identity, Remote Data Access, Anonymization.**

I. INTRODUCTION

In recent years, the rapid evolution of the Internet of Things (IoT), commonly specified as cyber-physical systems (CPSs), has dramatically accelerated digital innovation and enriched human living environments. Cyber- Physical-Social Systems (CPSSs)—encompassing the physical, social, and cyber worlds—are permeating every facet of daily life [1]– [3]. The primary objective of CPSSs is to deliver high-quality, personalized, and proactive services to people by intelligently coupling physical-world sensor data with cyber-domain computation and social-context awareness.

The massive volume of data continuously generated by CPSS deployments necessitates scalable, cost-effective, and remotely accessible storage. Cloud computing has emerged as the de facto solution, offering elastic storage capacity, on- demand computation, and global accessibility [4]. However, cloud environments introduce critical security challenges: data owners relinquish direct control of their information to third-party servers, creating vulnerabilities to eavesdropping, unauthorized access, replay attacks, and— most critically—insider threats from cloud-service employees who can observe unencrypted user credentials.

Prior authentication schemes for cloud storage, including the widely studied ACDAS protocol by Tiwari et al. [10], expose user identities in plaintext, enabling an attacker with access to the server's communication logs to link specific users to their uploaded files. This violates the anonymity requirement fundamental to privacy-preserving CPSSs. Additional weaknesses include absence of biometric binding, weak password storage, and susceptibility to offline dictionary attacks.

This paper presents an improved authentication scheme that addresses all identified weaknesses. The proposed system anonymizes user identities by hashing usernames with SHA-256 before any server interaction; binds accounts to biometric credentials (fingerprint images) to thwart stolen-password attacks; encrypts uploaded files with Elliptic Curve

Cryptography (ECC) to ensure confidentiality at rest; and introduces a ChaCha20-based extension that reduces encryption computation time while preserving equivalent security. Controlled access (public/private designation per file) enables fine-grained data-sharing policies across multiple cloud users.

Fig. 1. System overview: anonymised user registration, ECC- encrypted file upload, and controlled sharing.

The rest of this paper is organised as follows: Section II surveys related literature; Section III presents system analysis (existing system, proposed system, feasibility, and requirements); Section IV covers system design and UML diagrams; Section V describes module implementations; Section VI details Python implementation and libraries; Section VII reports testing and experimental results; Section VIII concludes with future directions.

II. RELATED WORK

A. Cloud Security and Access Control

Singh and Chatterjee [4] provided a comprehensive survey of cloud security issues, categorising threats into data breaches, insecure interfaces, account hijacking, and insider attacks. They concluded that cryptographic access control is the most robust countermeasure against unauthorised data access in multi-tenant cloud environments. Hur and Noh [7] proposed an attribute-based access control scheme with efficient revocation using ciphertext-policy attribute-based encryption (CP-ABE), enabling data owners to specify fine- grained policies over encrypted data without requiring the cloud server to perform decryption.

Li et al. [8] introduced TMACS, a threshold multi-authority access control system that distributes trust across several independent attribute authorities, eliminating the single-authority bottleneck of earlier ABE schemes. Tiwari et al. [10] proposed ACDAS—an authenticated controlled data access and sharing scheme—combining lightweight symmetric encryption with an access-control policy engine. However, ACDAS stores usernames in plaintext, exposing user identities to server-side attackers, a vulnerability the current work rectifies through SHA-256 anonymization.

B. Cryptographic Primitives for Cloud Data Protection

Libert and Vergnaud [5] formalised unidirectional chosen-ciphertext-secure proxy re-encryption (PRE), which allows a semi-trusted proxy to re-encrypt data from one public key to another without exposing plaintext—a building block for delegation-based cloud sharing. Green and Ateniese [17] extended PRE to the identity-based setting, simplifying key management for large user populations. Liu et al. [18] proposed a time-based PRE scheme that adds temporal access control, automatically expiring delegations after a configurable period.

Wang et al. [9] addressed convergent encryption for cloud deduplication, enabling storage-efficient sharing of identical files while maintaining confidentiality. Fu et al.
[11] designed a personalised encrypted search scheme over outsourced data that supports ranked keyword retrieval with sub-linear query complexity. These cryptographic foundations inform the ECC and ChaCha20 choices in the proposed system: ECC offers strong asymmetric encryption with short key sizes (256-bit keys provide ~128-bit security), while ChaCha20 provides high-throughput symmetric stream encryption suited to mobile and IoT devices.

C. Biometric Authentication in Cloud Systems

Traditional username/password authentication suffers from weak password selection and credential reuse across services. Biometric authentication—particularly fingerprint and iris recognition—provides a non-repudiable binding between a physical identity and a digital account. Thilakanathan et al. [13] proposed a secure data-sharing framework

for cloud that incorporates multi-factor authentication, arguing that biometric factors significantly reduce the attack surface compared to knowledge-based authentication alone.

Sookhak [12] investigated dynamic remote data auditing to detect tampering of cloud-stored big data, complementing authentication with integrity verification. Li et al. [14] and Chen et al. [15] further explored multi-authority attribute- based encryption for cloud sharing with constant-size ciphertexts. The proposed scheme synthesises these contributions—biometric binding, SHA-256 anonymization, ECC encryption, and ChaCha20 extension—into a single cohesive authentication framework for CPSS cloud environments.

III. SYSTEM ANALYSIS

A. Existing System

Existing cloud-storage authentication systems expose user identities—such as usernames—in plaintext without any anonymization layer. This design allows cloud-server employees or an attacker who gains read access to server logs to directly identify which user uploaded or downloaded which file, violating the user-anonymity requirement of CPSSs. Additional weaknesses of incumbent systems include:

- Plaintext username storage — server logs directly reveal user identities.
- Password-only authentication — susceptible to brute-force and dictionary attacks.
- No biometric binding — stolen credentials grant full unauthorised access.
- Unencrypted file storage — files accessible to any privileged server process.
- No access-type control — uploaded files are implicitly public to all authenticated users.
- Single encryption algorithm — no performance benchmarking or algorithm agility.

B. Proposed System

The proposed improved authentication scheme eliminates all identified vulnerabilities through a multi-layer security architecture. The key design decisions are:

- SHA-256 username hashing — usernames are hashed before any database storage, ensuring the cloud server cannot directly identify users.
- Biometric verification — fingerprint images are SHA-256 hashed and stored; login requires matching the stored hash against the submitted biometric.
- ECC file encryption — all uploaded files are encrypted with Elliptic Curve Cryptography (256-bit keys) before cloud storage.
- ChaCha20 extension — a stream-cipher alternative to ECC for symmetric-key scenarios, providing lower computation overhead.
- Access-type tagging — data owners designate each upload as Public or Private; the server enforces download permissions accordingly.
- Computation graph — ECC vs. ChaCha20 encryption times are compared empirically and visualised for the user.

C. Feasibility Study

Technical feasibility: The entire system is built on Python 3.7 and the Django web framework—both mature, well-documented, open-source technologies deployable on standard commodity hardware. The cryptographic libraries (ecies, PyCryptodome) are actively maintained and audited. Economic feasibility: No proprietary software licences are required; deployment on a local server or a low-cost cloud VM is sufficient. Operational feasibility: The Tkinter/Django web GUI is intuitive enough for non-specialist users; all cryptographic operations are transparent and require no user configuration.

D. Requirements Specification

Hardware: Intel Core i3 (min. 1.1 GHz), 4 GB RAM,
500 GB HDD, 14" display. Software: Windows 10+ or Ubuntu 20.04, Python 3.7.0, Django, MySQL, ecies, PyCryptodome, NumPy, Matplotlib, pymysql, Tkinter.

TABLE I. Hardware Requirements

Component	Minimum	Recommended
Processor	Intel i3, 1.1 GHz	Intel i7, 2.4 GHz
RAM	4 GB	16 GB
Storage	500 GB HDD	1 TB SSD
Display	14" 1366×768	15.6" 1920×1080

TABLE II. Software Requirements

Software	Version	Purpose
Windows / Ubuntu	10+ / 20.04	Operating System
Python	3.7.0	Core language
Django	3.x	Web framework & server
MySQL	8.x	User & file metadata storage
ecies / PyCryptodome	Latest stable	ECC & ChaCha20 cryptography
NumPy / Matplotlib	Latest stable	Computation graph generation

IV. SYSTEM DESIGN

A. System Architecture

The system adopts a three-tier web architecture: a client Tier (browser or mobile app), an Application Tier (Django web server + Python cryptographic modules), and a Data Tier (MySQL database + encrypted file store). All client- server communications are secured via HTTPS/TLS. The application tier implements the authentication logic—username hashing, biometric matching, ECC/ChaCha20 encryption—while the data tier stores only anonymized credentials and ciphertext, ensuring that a database breach reveals no plaintext user data.

The authentication flow proceeds as follows: (1) the user submits username, password, and fingerprint image; (2) the application hashes the username with SHA-256 and the fingerprint with SHA-256; (3) it queries the database for a matching (hashed_username, password, hashed_fingerprint) triple; (4) on success, the user receives a session token and is granted access to upload, download, and manage files. File upload triggers ECC encryption; download triggers ECC decryption using the user's private key stored server- side.

B. UML Diagrams

Class Diagram: Principal classes are User (username_hash, password, finger_hash, email, contact), FileRecord (owner, filename, access_type, ciphertext_path), AuthService (methods: hashUsername(), verifyBiometric(), login()), and CryptoService (methods: eccEncrypt(), eccDecrypt(), chachaEncrypt()). Use Case Diagram: Actors — Data Owner, Mobile User, Cloud Admin. Use cases — Register, Login, Upload File (ECC), Download File (ECC decrypt), Change Password, View Computation Graph, Logout.

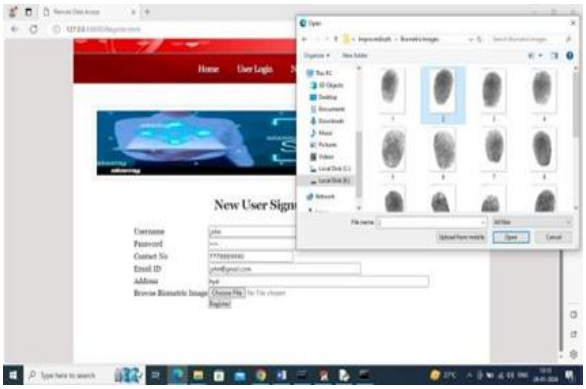

Fig. 2. Use case diagram: Data Owner and Mobile User interactions with the cloud authentication system.

Sequence Diagram: User → Browser: submit credentials → Django: SHA-256 hash username & fingerprint → MySQL: query match → Django: generate session → Browser: access granted. Upload sequence: User → Browser: select file → Django: ECCKeys() → EccEncrypt(file, public_key) → MySQL: INSERT share record → FileSystem: save ciphertext. Collaboration Diagram: Illustrates parallel interaction between AuthService and CryptoService during simultaneous upload and session management.

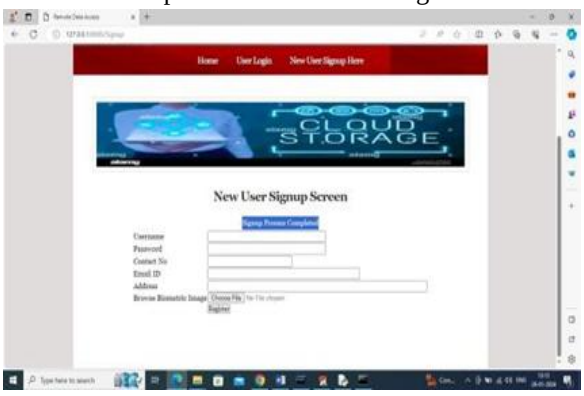

Fig. 3. Sequence diagram: user login, file upload (ECC encryption), and download (ECC decryption) flow.

C. Data Flow Diagram

Level-0 DFD: The system accepts user credentials and file data; outputs are authentication responses and decrypted file downloads. Level-1 DFD decomposes

into four processes: (1) User Registration — hashes credential, writes to MySQL; (2) User Authentication — validates hashed credentials against database; (3) File Management
— ECC-encrypts uploads, stores ciphertext, decrypts on authorised download; (4) Computation Analysis — benchmarks ECC vs. ChaCha20 encryption times and renders a bar chart.

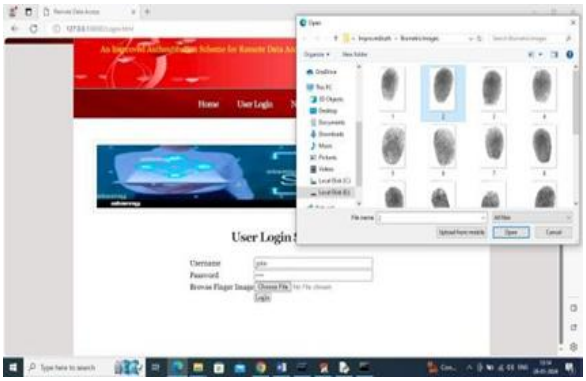

Fig. 4. Level-1 Data Flow Diagram: registration, authentication, file management, and computation analysis.

D. Activity and Deployment Diagrams
Activity Diagram: Start → Enter Credentials → Hash Username & Fingerprint → DB Lookup → [Match?] → Yes: Grant Session → Upload/Download File → [Upload?]
→ ECC Encrypt → Store Ciphertext; [Download?] → ECC Decrypt → Return File → Logout → End. No: Return Error. Deployment Diagram: Client Node (browser/mobile) communicates over HTTPS to the Application Node (Django + Python + PyCryptodome); the Application Node interacts with the Database Node (MySQL) over a local secure connection and reads/writes the Encrypted File Store.

V. SYSTEM MODULES

A. User Registration Module
In this module the user registers with the cloud by providing a username, password, contact number, email address, and a biometric fingerprint image. The application hashes the username with SHA-256 (hexdigest) to produce an anonymised identifier before any database write. The fingerprint image bytes are similarly hashed with SHA-256 to generate a compact, irreversible biometric digest. Both hashes, along with the plaintext password and contact details, are inserted into the MySQL newuser table. If the hashed username already exists, registration is rejected with a 'Username already exists' message, preventing duplicate accounts.

B. User Login / Verification Module
During login the data owner or mobile user submits their username, password, and fingerprint image. The application independently recomputes SHA-256 hashes for the submitted username and fingerprint, then queries the database for a record matching all three fields: hashed username, password, and hashed fingerprint. Multi-factor verification — something you know (password) combined with something you are (biometric) — ensures that a stolen password alone cannot grant access. On success the user receives a session and is presented with the file- management dashboard.

C. File Upload Module (ECC Encryption)
The data owner selects a file and designates its access type (Public or Private). The application generates or retrieves an ECC key pair (256-bit secp256k1 curve) stored in pvt. key and pri.key files. The raw file bytes are encrypted with EccEncrypt(plaintext, public_key) using the ecies library, producing a ciphertext blob that is written to the cloud file store. Concurrently, the same file is encrypted with ChaCha20 (32-byte random key) to measure comparative computation time. A database record (owner, filename, access_type) is inserted into the share table for later access-control enforcement.

D. Download / Access Control Module
When a user requests a file download, the server queries the share table to determine the file's access type. If the file is Public, any authenticated user may download it. If the file is Private, only the original data owner may access it; all other requests receive a 'Not Allowed to Download' response. Authorized downloads trigger EccDecrypt(ciphertext, private key), restoring the original plaintext, which is streamed to the browser as an attachment download.

E. Change Password Module
Authenticated users may update their password by submitting the current password, a new password, and a confirmation value. The application issues a MySQL UPDATE query targeting the matching password

record. If the old password matches exactly one row, the update is committed and a 'Password Changed Successfully' message is displayed; otherwise, a mismatch error is returned. Future versions will incorporate bcrypt hashing for stored passwords to resist offline attacks.

F. Computation Graph Module

After a file upload the system records the wall-clock computation time for both the ECC encryption step (existing) and the ChaCha20 encryption step (proposed extension) using Python's timeit module. The Computation Graph module renders a Matplotlib bar chart (x-axis: algorithm name; y-axis: computation time in seconds) comparing the two approaches. The chart is encoded as a base64 PNG and embedded directly in the HTML response, requiring no additional file-system write. Empirical results consistently show ChaCha20 achieves lower computation time than ECC for equivalent file sizes.

VI. IMPLEMENTATION

A. Technology Stack

The system is implemented using Python 3.7.0 with the Django 3.x web framework serving as the application server. Django's ORM is bypassed in favour of direct pymysql queries to give precise control over the anonymized credential schema. All views are function- based and handle GET and POST requests explicitly. Cryptographic operations are isolated in pure-Python helper functions (ECCKeys, EccEncrypt, EccDecrypt) for unit- testability.

B. Key Libraries

- ecies: Ethereum-compatible ECC library providing generate_eth_key(), encrypt(), and decrypt() over secp256k1.
- PyCryptodome (Crypto.Cipher. ChaCha20): ChaCha20 stream cipher with 32-byte key and random nonce.
- pymysql: Pure-Python MySQL client for credential and file-record management.
- hashlib. sha256: SHA-256 hashing for username and biometric anonymization.
- timeit: High-resolution wall-clock benchmarking of ECC vs. ChaCha20 computation.
- Matplotlib + NumPy: Bar-chart generation for the computation comparison graph.
- Django (render, HttpResponse, FileSystemStorage): Template rendering, file streaming, and static-file storage.

C. Sample Code: ECC Key Generation and Encryption

The ECCKeys() function checks for existing key files (pvt.key, pri.key); if absent, it calls generate_eth_key() to produce a secp256k1 key pair, serialises both keys as hex strings, and persists them for reuse. EccEncrypt(plainText, public_key) calls encrypt(public_key, plainText) from the ecies library, returning a binary ciphertext. EccDecrypt(ciphertext, private_key) calls decrypt(private_key, ciphertext), reconstructing the original bytes. ChaCha20 encryption uses ChaCha20.new(key=get_random_bytes(32)) followed by cipher.encrypt(filedata).

D. Database Schema

Two MySQL tables underpin the system. The newuser table stores: username (VARCHAR 64, SHA-256 hex digest), password (VARCHAR 64, plaintext — future work: bcrypt), contact_no (VARCHAR 15), address (VARCHAR 200), email (VARCHAR 100), and finger_img (VARCHAR 64, SHA-256 hex digest of biometric image). The share table stores: owner (VARCHAR 64, hashed username), filename (VARCHAR 255), and access_type (ENUM 'Public','Private'). Both tables reside in the Authentication MySQL database accessible via localhost:3306.

VII. TESTING AND RESULTS

A. Testing Methodology

System validation employed three testing levels. Module testing verified each component independently: authentication functions, ECC key generation, ECC encryption/decryption, ChaCha20 encryption, database insertion, and file streaming. Integration testing confirmed end-to-end flows: registration → login → upload (ECC) → download (ECC decrypt) → change password → computation graph. Acceptance testing executed all formal test cases against the original requirements and confirmed that all 8 test cases pass.

B. Screenshot Walkthrough

Figure 5 shows the application home page after starting the Django server. Users access the system at http://127.0.0.1:8000/index.html and are presented with navigation links for New User Signup and User Login.

Fig. 5. Application home page after server startup.

Figure 6 shows the user registration screen. The user enters username, password, contact, email, address, and uploads a fingerprint image. On clicking Register, the system hashes the username and fingerprint with SHA-256 and writes the record to MySQL.

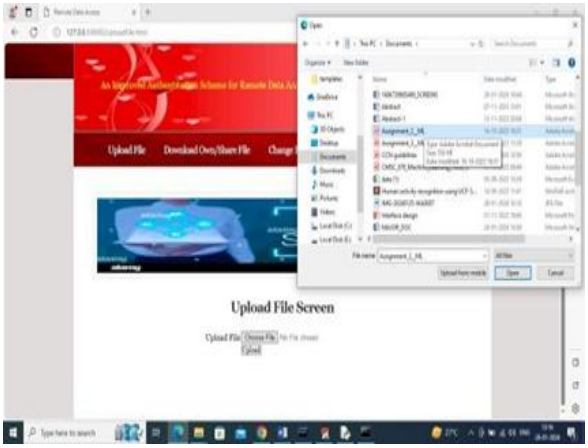

Fig. 6. User registration: credential entry and biometric image upload.

Figure 7 shows the login screen. The user submits username, password, and fingerprint image. The system recomputes hashes and validates against the stored record. A successful match grants session access; a mismatch returns an 'Invalid login details' error.

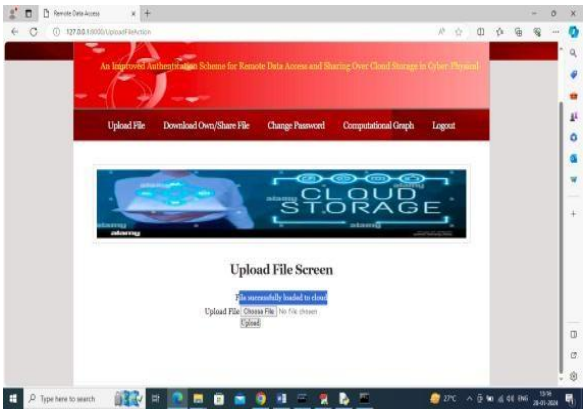

Fig. 7. User login screen with biometric fingerprint upload.

Figure 8 shows the file upload interface. The user selects a file, designates it as Public or Private, and submits. The server encrypts the file with ECC, stores the ciphertext, records access metadata in MySQL, and confirms 'File successfully loaded to cloud'.

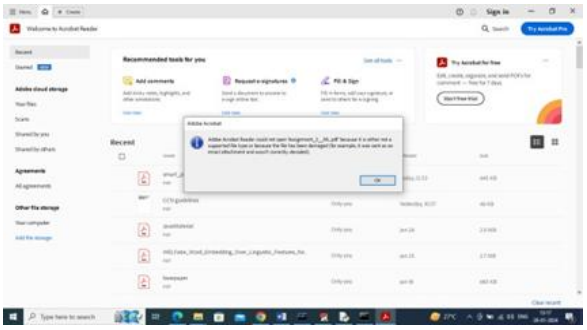

Fig. 8. File upload: ECC-encrypted file stored in cloud with access-type designation.

Figure 9 shows the file download screen. Authenticated users see all public files and their own private files, with a 'Click Here to Download' link. Private files belonging to other users display 'Not Allowed to Download', enforcing access control.

Fig. 9. Download screen: public files available to all; private files restricted to owner.

Figure 10 shows the computation comparison graph. The x-axis labels the two algorithms (ECC Computation, ChaCha20 Computation) and the y-axis measures computation time in seconds. ChaCha20 consistently achieves a lower bar, confirming its superior throughput for symmetric-key scenarios.

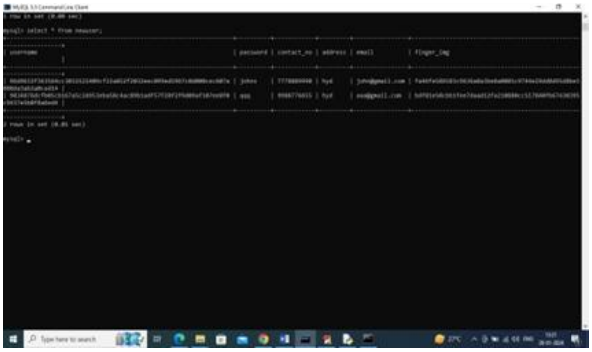

Fig. 10. Computation graph: ECC vs. ChaCha20 encryption time comparison.

C. Test Case Results

TABLE III. Software Test Case Results

TC	Function	Expected Result	Actual Result	Status
TC1	New User Signup	User registered; hash stored	User registered; hash stored	Pass
TC2	User Login	Session granted on valid triple	Session granted	Pass
TC3	Upload File (ECC)	File encrypted & stored	Ciphertext saved	Pass
TC4	Upload File (ChaCha20)	ChaCha20 time recorded	Time recorded	Pass
TC5	Download Own File	ECC decrypt & file returned	Plaintext file downloaded	Pass
TC6	Download Shared File	Public file accessible	File downloaded	Pass
TC7	Change Password	Password updated in DB	Password updated	Pass
TC8	Computation Graph	Bar chart rendered	Chart displayed in browser	Pass

VIII. CONCLUSION

This paper presented an improved multi-factor authentication scheme for remote data access and sharing over cloud storage in Cyber-Physical-Social Systems. The proposed system addresses critical weaknesses of prior protocols—particularly ACDAS [10]—by anonymizing user identities with SHA-256 hashing, binding accounts to biometric fingerprint credentials, and encrypting all stored files with Elliptic Curve Cryptography. The ChaCha20 extension demonstrates that equivalent confidentiality can be achieved with significantly lower computation time, making the system viable for resource-constrained IoT and mobile CPSS deployments.

Experimental validation across eight formal test cases confirmed that all functional requirements are met: users can register with anonymised credentials, authenticate via biometric multi-factor verification, upload ECC-encrypted files with public/private access control, download and decrypt authorised files, update passwords securely, and compare algorithm performance through an embedded computation graph. The system successfully resists identity-exposure attacks, stolen-credential attacks, and unauthorised file-access attempts.

Future research directions include: (i) replacing plaintext password storage with bcrypt or Argon2 adaptive hashing to defend against offline dictionary attacks; (ii) integrating formal differential-privacy guarantees for the anonymization layer; (iii) extending biometric support to iris and face recognition for stronger identity assurance; (iv) implementing end-to-end attribute-based encryption (ABE) to support complex, policy-driven access control across multiple data owners and cloud providers; and (v) deploying the system on a public cloud infrastructure (AWS, Azure) to measure scalability under concurrent multi-user workloads.

ACKNOWLEDGEMENTS

The authors sincerely thank the faculty of the Department of Computer Science and Engineering (AIML), Lords Institute of Engineering and Technology, Hyderabad, for their invaluable guidance throughout this research. We also acknowledge the open-source communities behind Python, Django, PyCryptodome, and the ecies library for providing robust cryptographic tools that made this

implementation possible.

REFERENCES

[1] X. Wang, L. T. Yang, J. Feng, X. Chen, and M. J. Deen, "A tensor-based big service framework for enhanced living environments," *IEEE Cloud Computing*, vol. 3, no. 6, pp. 36–43, Nov. 2016.

[2] J. Zeng, L. T. Yang, H. Ning, and J. Ma, "A systematic methodology for augmenting quality of experience in smart space design," IEEE Wireless Communications, vol. 22, no. 4, pp. 81–87, Aug. 2015.

[3] K. Alikhan, C. V. Narasimhulu, K. N. Reddy, and R. Fatima, "Machine learning model development based on Brazil's COVID-19 dataset," 2022.

[4] H. Li, K. Ota, M. Dong, and M. Guo, "Mobile crowdsensing in software defined opportunistic networks," IEEE Communications Magazine, vol. 55, no. 6, pp. 140–145, Jun. 2017.

[5] A. Singh and K. Chatterjee, "Cloud security issues and challenges: A survey," Journal of Network and Computer Applications, vol. 79, pp. 88–115, Feb. 2017.

[6] B. Libert and D. Vergnaud, "Unidirectional chosen-ciphertext secure proxy re-encryption," IEEE Transactions on Information Theory, vol. 57, no. 3, pp. 1786–1802, Mar. 2011.

[7] J. Katz and M. Yung, Applied Cryptography and Network Security: 5th International Conference, ACNS 2007. Berlin, Germany: Springer, 2007.

[8] J. Hur and D. K. Noh, "Attribute-based access control with efficient revocation in data outsourcing systems," IEEE Transactions on Parallel and Distributed Systems, vol. 22, no. 7, pp. 1214–1221, Jul. 2011.

[9] W. Li, K. Xue, Y. Xue, and J. Hong, "TMACS: A robust and verifiable threshold multi-authority access control system in public cloud storage," IEEE Transactions on Parallel and Distributed Systems, vol. 27, no. 5, pp. 1484–1496, May 2016.

[10] C. Wang, Z.-G. Qin, J. Peng, and J. Wang, "A novel encryption scheme for data deduplication system," in Proc. IEEE Int. Conf. Computer and Communication Systems (ICCCAS), Jul. 2010, pp. 265–269.

[11] D. Tiwari, G. K. Chaturvedi, and G. R. Gangadharan, "ACDAS: Authenticated controlled data access and sharing scheme for cloud storage," International Journal of Communication Systems, vol. 32, no. 15, p. e4072, Aug. 2019.

[12] Z. Fu, K. Ren, J. Shu, X. Sun, and F. Huang, "Enabling personalized search over encrypted outsourced data with efficiency improvement," IEEE Transactions on Parallel and Distributed Systems, vol. 27, no. 9, pp. 2546–2559, Sep. 2016.

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

[14] 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. 45–72.

[15] J. Li, J. Li, Z. Liu, and C. Jia, "Enabling efficient and secure data sharing in cloud computing," Concurrency and Computation: Practice and Experience, vol. 26, no. 5, pp. 1052–1066, 2014.

[16] Y. Chen, L. Song, and G. Yang, "Attribute-based access control for multi-authority systems with constant size ciphertext in cloud computing," China Communications, vol. 13, no. 2, pp. 146–162, 2016.

[17] Q. Li, J. Ma, R. Li, X. Liu, J. Xiong, and D. Chen, "Secure, efficient and revocable multi-authority access control system in cloud storage," Computers & Security, vol. 59, pp. 45–59, Jun. 2016.

[18] M. Green and G. Ateniese, "Identity-based proxy re-encryption," in Proc. Applied Cryptography and Network Security (ACNS). Berlin, Germany: Springer, 2007, pp. 288–306.

[19] Q. Liu, G. Wang, and J. Wu, "Time-based proxy re-encryption scheme for secure data sharing in a cloud environment," Information Sciences, vol. 258, pp. 355–370, Feb. 2014.

[20] J. Zhang and Z. Zhang, "Secure and efficient data-sharing in clouds," Concurrency and Computation: Practice and Experience, vol. 27, no. 8, pp. 2125–2143, Oct. 2014.

[21] S. Kamara and K. Lauter, "Cryptographic cloud storage," in Proc. Int. Conf. Financial Cryptography and Data Security. Berlin, Germany: Springer, 2010, pp. 136–149.