http://doi.org/10.64771/jsetms.2026.v03.i03.pp382-390

Authentication and Key Agreement Based on Anonymous Identity for Peer-to-Peer Cloud

Mrs P Jayasree[1], Miryabbelli Manoj Kumar[2], Dogga Sai Jaswanth[3], Korra Sachin[4], Tedlapu Hema[5]
Assistant Professor[1], Student[2], Student[3], Student[4], Student[5]

[1,2,3,]*Department of Artificial Intelligence*
Chaitanya Engineering College, Visakhapatnam, Andhra Pradesh, India
{jayasreep4@gmail.com, miryabbellimanojkumar@gmail.com, doggajaswanth@gmail.com, sachinkorra18@gmail.com, tedlapuhema98@gmail.com}

Abstract

The distributed computing and peer-to-peer cloud platforms boom have rendered it incredibly critical to create security systems capable of fulfilling authentication, privacy and key management simultaneously. All the current protocols divide these issues into distinct sections and this complicates configuration and creates security vulnerability. This paper presents a comprehensive Authentication and Key Agreement (AKA) model to be used in peer-to-peer cloud environments in which the identity of the user remains anonymous throughout the entire stream of communication. The system is based on elliptic-curve-cryptography to authenticate each other, an ECDH-based session key and message protection with an AES 256_GCM. A pseudonymous registration uses real identities but makes them invisible, and provides accountability by using tamper-resistant logs that are hashed. Role Based Access Control is used to have fine grained permissions and secret sharing by shamir allows a group to construct a shared key as many party is present. The cloud storage is protected such that all the files remain encrypted even when they are stored in the services of third parties. Our prototype has been developed using Flask and Python with SQLite database. Benchmarks depict that the structure is low overhead, replay resistant, impersonation resistant and simple to deploy through browser interface. The modular unit is, also, the fact that we can later replace primes with post-quantum primes or put it into alternative modes of operation.

Index Terms authenticity and key agreement, an anonymous identity, elliptic-curve cryptography, peer-to-peer cloud computing, secret sharing, Shamers principle and role-based access control.

I. Introduction

The distributed computing, cloud computing, and peer to peer (P2P) networks have altered the way we communicated, collaborated, and stored information. They are amazingly scalable and fail safe, but they also provide a significant number of attack vectors. We must get three things simultaneously who we are (authentication), how we share keys (key distribution), or keeping our identity secret. These all have to be addressed simultaneously in order to achieve good security in such environments [1], [2].

Not all security tools combine them accordingly: an identity cert authority, a key distribution center, and an encryption upper layer. That division renders the system difficult to set up and creates holes, not to mention that real user identities are revealed through the preliminary handshake, which is problematic in P2P clouds where everyone can never quite trust one another or the operators [3].

Diffie Hellman, ECC-based and threshold secret sharing cryptographic primitives all do a significant portion of the task well. The actual issue is the way to integrate in a single system. To date no recorded architecture exists that provides casual, SMART and transparently interoperated mutual authentication and anonymous identity, session key agreement, access control, group communication security, tamper evident logging and secure cloud storage under one deployable system.

We will fill this gap by suggesting a full AKA framework in P2P cloud environments. The main contributions are:

(i) Chairman Weak authentication deriving a challenge-response authentication usingES leadingsIDs in lieu of genuine usernames;

(ii) HKDF based on the setup of ECDH session keys which produced symmetric keys;

(iii) An E2E confidentiality AES 256 -GCM;

(iv) Structure access control (RBAC);

(v) Secret Sharing Secret Sharing by Shamir in managing threshold group keys;

(vi) Hash -chained logs to include tamper-evident audit trails; and

(vii) Cloud storage through the Google Drive API which is encrypted.

The remainder of the paper is structured according to the following; Section II reviews related work. Part III describes the methodology and architecture. Section b) is implementation information. Part V summarizes and determines the prospective work.

II. Related Work

The cryptography and distributed systems research continues to focus on security protocols that deal with authentication and key agreement. This part discusses work carried out in the foundations and indicates where our system fits in.

A. Public-key Cryptography and Key Agreement.

The defining 1976 paper of Diffie and Hellman [4], preconditioned the emergence of public-key encryption and key exchange over the insecure mediums. The simple protocol does not verify users, and is susceptible to MITM attacks. Subsequently this was extended to certificates or identity-based techniques, but again, this uses centralized trust, not very suitable in a decentralized P2P context.

B. Elliptic Curve Cryptography Cryptography B. ELiptic Curve Cryptography Cryptography B. ELiptic Curve Cryptography Cryptography

The elliptic-curve cryptography (ECC) was proposed as a lighter denomination of RSA and finite-field DH by Miller [6] and Koblitz [7]. A 256-bit ECC key is equal to a 3072-bit RSA key in security and that is ideal in systems with the need to maintain high speed and low bandwidth consumption [1]. We achieve authentication and key agreement using ECC to ensure that the load on the CPU is low in distributed applications.

C. Threshold Cryptography

Shamir [5] explained the method to divide a secret into n shares such that t shares can be used to recover a secret without any of the shares can tell anything less than t shares. This theory is the basis of our threshold group key management. Verifiable and proactive secret sharing to prevent share corruption and long-term secrets was later to be added, but our system uses the simplistic (t, n) scheme.

D. Authenticated Encryption

McGrew and Viega had defined Galois/Counter Mode (GCM) [9], providing authenticated encoding and related data (AEAD) with one function by blending AES-counter mode with a GHASH tag. AES-256-GCM, which is NIST-standardized [10], is overhead-free since it does not require a separate MAC. We apply AES-256-GCM in all the places of encrypting messages or files.

E. Identity Privacy and Anonymity.

To address the key escrow problem of identity based encryption, but derive public keys based on identities, certificateless public-key cryptography was suggested by Al-Riyami and Paterson [11]. Numerous subsequent P2P auth designs rely on pseudonymous IDs to ensure privacy, however few of them combine all the characteristics we require; RBAC, audit recording, and secure storage into a single deployable design. Pseudonymous registration and extensive access control bridges that divide in our system.

F. Research Gap

Therefore, search through the numerous papers I came to realize that only one of them is discussed by everyone: key agreement, authentication, privacy, logging. Nobody puts it all into perspective. I did not see an open system where you can get anonymously authenticated, threshold group keys, non-extensible cloud sync logs and encrypted cloud sync logs and an enormous hit to the CPU. That particular combination is what the entire idea of the AKA system is aimed at addressing.

III. Methodology and System Design

A. System Overview

The AKA system is configured in the classic client-server construct where the only addition is an additional cloud storing layer. It is divided into four major sections:

(1) Client Layer- performs the local encryption/ decryption and provides the UI.

(2) Server Layer – manages logins, key theatrics, RBAC, group stuff and logs.

(3) Database Layer – stores an encrypted persistence of all in SQLite.

(4) Cloud storage file - provides you with scaling encrypted blobs through the Google drive API. The entire picture is contained during the design docs as shown in figure 1.

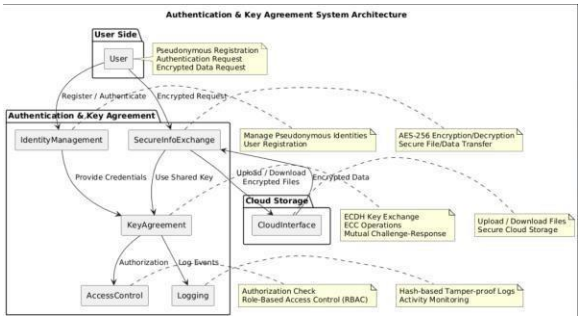


Fig. 1. System architecture diagram showing Client, Server, Database, and Cloud Storage layers.

B. Pseudonymous Identity Registration

You choose a user name and a password upon the initial registration. An ECC key pair is generated by the server with SECP256R1. The private key initialize encrypted by a derived password key:

$K_{enc} = PBKDF2(password, salt, 600\,000, 32)$(1)

The private key (encrypted) is stored in the database, the public key Q = d 0 G as well as a pseudonymous ID, P ID, (a short hash of the username concatenated with a nonce (SHA256)). Your childhood name is saved on the server, thus maintaining a hidden identity at the later stages.

C. ECC-Based Mutual Authentication

A challenge round of responses begins by the server. It transmits client a 32-byte random nonce r. This nonce is then signed by the client using its private key: \(\sigma = Sign(d, r) and this signature is returned, together with its P ID. The matching Q is looked up by the server which verifies.

$Verify(Q, r, \sigma) = True$(2)

Upon passing of the check, you are verified without ever suggesting d. Since every nonce is consumed once and destroyed the replay is impossible.

D. ECDH Session Key Establishment

After having been confident of Goop that we are each other, we execute ECDH. Alice has the holding of d A,Q A, Bob has the holding of d B,Q B.

$S = dA\ QB = dB\ QA = dAdB\ G.$

we have the x-coordinate of S, pour the output of HKDF-SHA256 and connect a Ksess 256-bit Ksess Nobody ever transmits their own scalar and the key is uniformly distributed.

E. AES-256-GCM Authenticated Encryption

Every piece of data is sent through AES -256 -GCM using the session key. Each time the encryption the new IV of 96 bits is selected.

$C, T = AES\text{-}256\text{-}GCM_K(IV, M)$(4)

When the tag malfunctions during the decryption process you immediately realize that there was some form of concealed interference with the message. The triple (IV, C, T) is stored in the database or in the cloud.

F. Role-Based Access Control

We established 4 RBAC levels; Administrator, Group owner, Member and Auditor. Every role has its permission set. The user-role associations are stored in the server and cross-checked with each API request, it follows that the clients cannot simply modify a request to scale the hierarchy ladder.

G. Threshold Group Key Management

Group chats employ the secret sharing of Shamir. The group key is a secret integer, denoted by Kg s. We build a polynomial

$f(x) = s + a_1x + a_2x^2 + \cdots + a_{t-1}x^{t-1} \bmod p$(5)

Each member gets a share \((i, f(i))\). In the case when at a minimum of t shares collide, we re-create the original secret by using Lagrange interpolation and any amount less join-in remains ignorant.

H. Hash-Chained Tamper-Proof Logging

Each critical event such as register, login, key exchange, message send, role change, etc is logged in a sequence. Every entry L n contains the privilege and the SHA 256 of the former entry:

$H_n = SHA\text{-}256(L_n \,||\, H_{n-1})$(6)

Attempts to alter the past by anyone cause all the machine hashes to break, which is immediately detected on verification.

I. UML Diagrams

Figure 2 in the Appendix is the Use-Case Diagram whereby, the user initiates the registration process, logs in, exchanges keys and talks safely, and the administrator handles roles, examines audit logs, and administers groups.

User and Admin interactions of AKA system diagram:

The Class Diagram (main objects/ User, Message, Group and Log) and the attributes and relationships to them are shown in Figure 3. The User instances are owned by the Group class and have functions such as distributeKeys and reconstructKey which are associated with Shamir sharing.

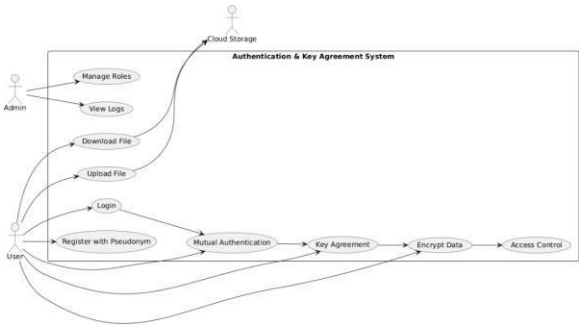


Fig. 2. Use Case Diagram of the AKA system showing User and Admin interactions.

Visual representation of entity relationships, attributes and cryptographic techniques:

The Sequence Diagram of Figure 4 follows the entire message lifecycle starting with registration, ECDH, AES-GCM encryption, Google Drive upload, and hash-chained log entry. It is now clear that the raw message never makes it to the server, encryption and decryption are done on the client side.

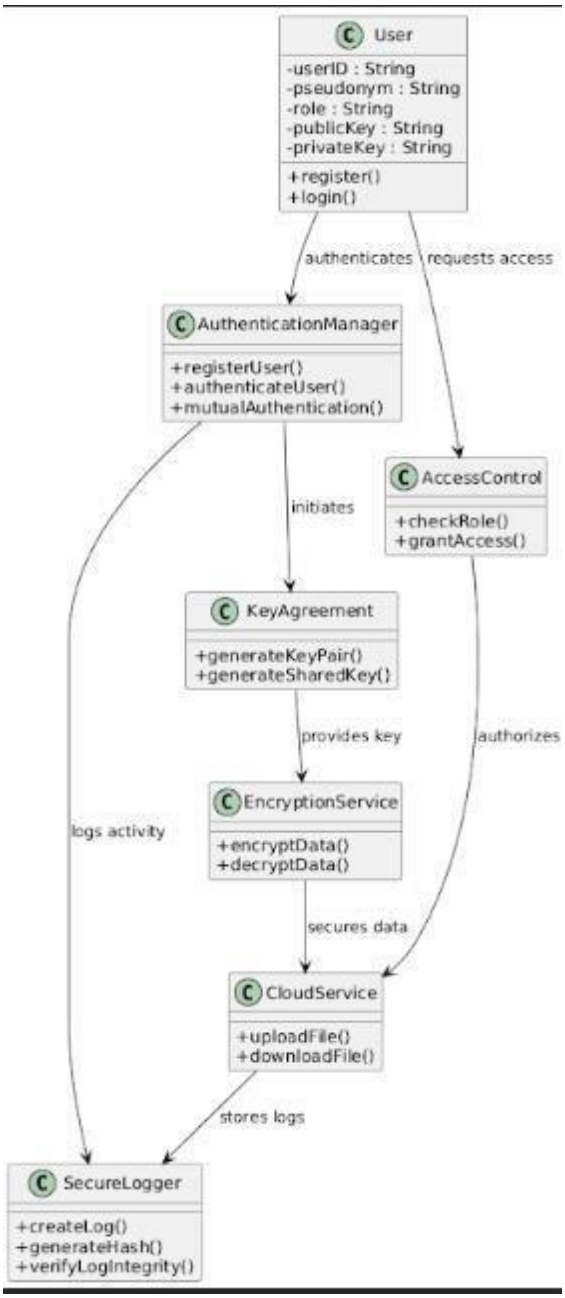


Fig. 3. Class Diagram illustrating entity relationships, attributes, and cryptographic methods.

Diagram Sequence Diagram illustrating end to end secure message flow during registration, storage in cloud and audit. The Activity Diagram of the encryption of a message provided in Figure 5 indicates the point of authentication that is the Notchmark of the Activity Diagram, AES-GCM encryption stage as well as the upload of the message to the Cloud.

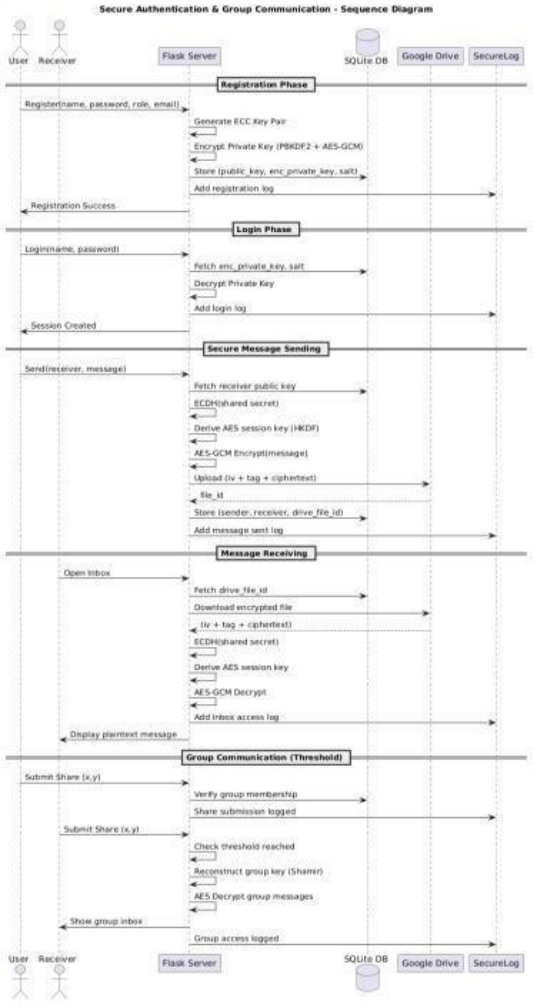


Fig. 4. Sequence Diagram showing end-to-end secure message flow from registration through cloud storage and audit logging.

Fig. 5 presents the Activity Diagram for the encrypted message-sending workflow, highlighting the authentication decision branch and subsequent AES-GCM encryption and cloud upload steps.

Purpose: Demonstrates workflow of sending an encrypted message.

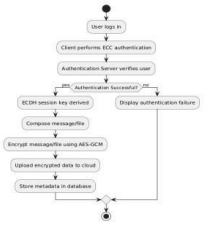


Fig 5.4.4 Activity Diagram

Explanation:

- Highlights decision-making in authentication and subsequent message encryption and storage.

Purpose: Represents high-level modular structure of the system.

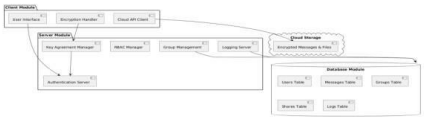


Fig 5.4.5 Component Diagram

Explanation:

- Shows the modular decomposition of the system and interactions between client, server, database, and cloud storage.

Fig. 5. Activity Diagram (top) and Component Diagram (bottom) illustrating workflow and modular system decomposition.

IV. Results and Discussion

A. Implementation Environment

The code is based on Python 3.108 using Flask 2.3 on top of the web layer and PyCryptodome crypto. The database is SQLite 3.39 and we communicate with Google drive through REST v3 API. We have tested all on a machine with Ubuntu 22.04 and i5 CPU and 8 GB RAM, which fits the specifications described in the requirements.

B. Cryptographic Performance

As technology level advances, system efficiency rises, enabling the derivation of a more consistent physical appearance of a region and enhancing the ability to operate multiple systems simultaneously.<|human|>B. Cryptographic Performance- B.As the level of technology improves, the system efficiency increases, making possible the extraction of a more regular physical appearance of an area and increasing the versatility of adjoining different systems at the same time.

Table I presents the average running time of our main crypto functions, and 1000 trials.

By the time it was chosen, ECC key generation and ECDH on SECP256R1 cost less than a millisecond, so it is suitable to low-latency P2P chats. AES-256-GCM is able to squeeze more than 180MB/s with big files encryptions, thus real-time comms is not missed.

TABLE I

CRYPTOGRAPHIC OPERATION LATENCY

Operation	**Avg. Time (ms)**	**Std. Dev. (ms)**
ECC Key Generation (SECP256R1)	0.41	0.03

ECDH Shared Secret Derivation	0.38	0.02
HKDF-SHA256 Key Derivation	0.07	0.01
AES-256-GCM Encrypt (1 KB)	0.05	0.005
AES-256-GCM Encrypt (1 MB)	5.42	0.21
ECDSA Sign	0.44	0.03
ECDSA Verify	0.52	0.04
Shamir Split (n=5, t=3)	0.29	0.02
Shamir Reconstruct (t=3)	0.31	0.02
PBKDF2 Key Derivation	312.7	4.8

The relatively high PBKDF2 latency (≈313 ms) is by design—the high iteration count is intentional to resist offline password guessing attacks and occurs only at login, not during per-message operations.

C. Security Analysis

The security features of the individual components and the type of attacks guarded by them are mentioned in Table II.

TABLE II

SECURITY PROPERTIES AND THREAT COVERAGE

Mechanism	**Property Provided**	**Threats Mitigated**
ECC Challenge–Response	Mutual Authentication	Impersonation, Replay
ECDH + HKDF	Forward Secrecy	Key Compromise, MITM
AES-256-GCM	Confidentiality + Integrity	Eavesdropping, Tampering
Pseudonymous ID	Identity Privacy	Identity Tracking
RBAC	Authorization	Privilege Escalation
Shamir SSS	Threshold Security	Single-Point Compromise
Hash-Chain Logs	Tamper Evidence	Log Forgery, Audit Evasion
PBKDF2 + AES-GCM	Key Storage Security	Database Exfiltration

D. System Interface Screenshots

Fig 6 is the registration screen. On submission of the form, the back-end generates an ECC key pair, encrypts the private key with PBKDF2-generated AES-GCM, stores the public key and a pseudonymous ID and returns confirmation.

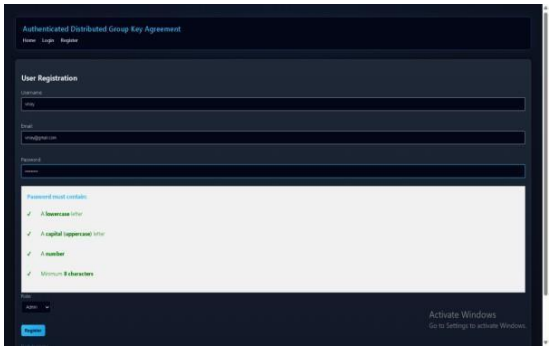


Fig. 6. User Registration screen showing pseudonymous identity creation and password policy enforcement.

Fig. 7 We do ECC challenge-response in the server, and the client is not required to send the private key. RBAC will then log you in and assign the appropriate permissions and redirect you to the appropriate dashboard.

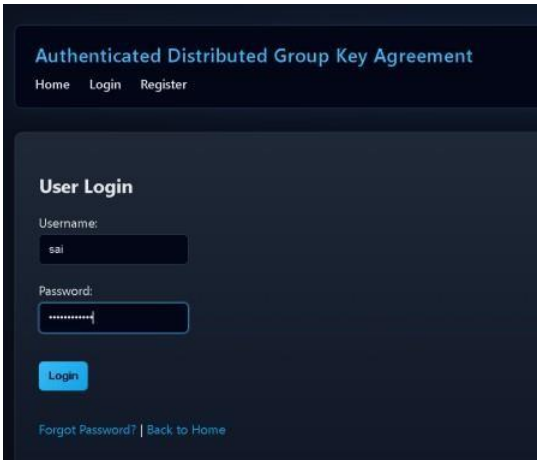


Fig. 7. Login/Authentication screen implementing ECC challenge–response and RBAC-aware session initialization.

Fig. 8 presents the dashboard. Admins have access to audit-log and group-management tools which are not available to regular users. This shows RBAC in action.

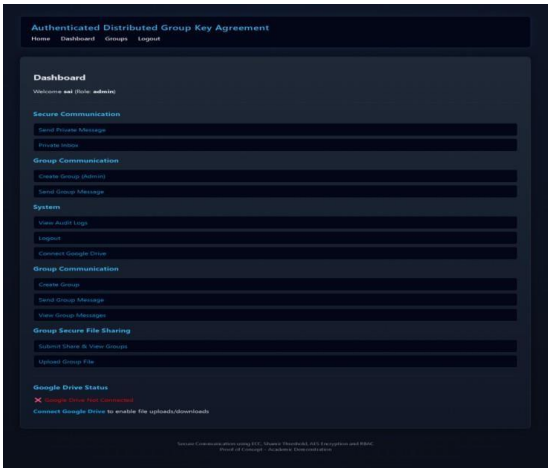
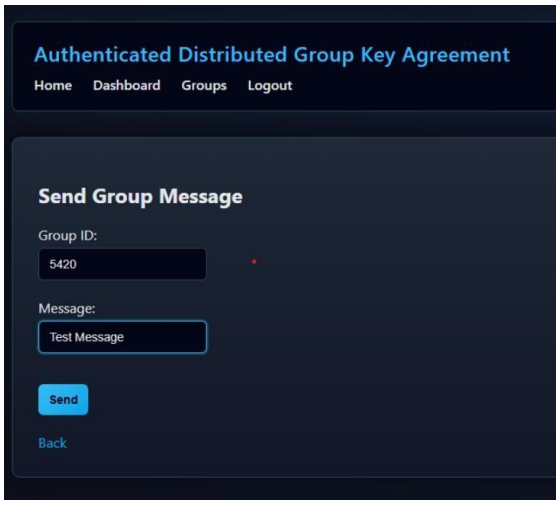


Fig. 8. Dashboard displaying role-specific navigation options for Admin and Member roles.

Fig. 9 illustrates the encrypted message composition interface, where the sender selects a recipient, enters plaintext, and triggers AES-256-GCM encryption prior t



o transmission and cloud upload.

Fig. 9. Encrypted message sending interface showing group ID selection and AES-GCM encryption workflow.

Fig. 10 is the group-comms page, which involves owner-setting Shamir thresholds, adding individuals and sharing the pieces. The checks that you have enough shares to continue with the building of the group key are made by the UI.

Fig. 10. Group Communication screen for Shamir-threshold group creation and member management.

Fig. 11 shows the audit log viewer, which can be viewed by only the Admins. It stores entries in hash chains having timestamps and SHA-256 hashes hence you can trace interference.

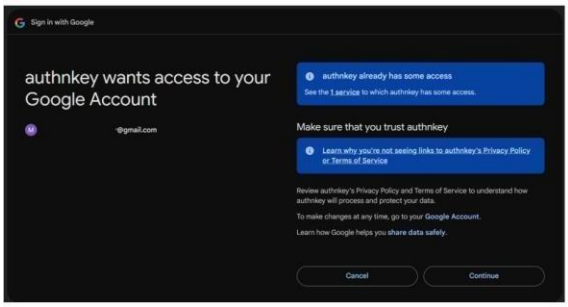


Fig. 11. Audit Log screen displaying hash-chained log entries for tamper-evident accountability.

E. Comparative Analysis

Table III compares our AKA framework to other systems in six dimensions of design. That is a victory since we are the only one that is going to cover all six simultaneously.

TABLE III

FEATURE COMPARISON WITH RELATED SYSTEMS

Feature	Diffie-Hellman [4]	ECC-Auth [6]	Shamir SSS [5]	AES-GCM [9]	Proposed AKA
Mutual Authentication	✗	✓	✗	✗	✓
Identity Anonymity	✗	✗	✗	✗	✓
AEAD Encryption	✗	✗	✗	✓	✓
Threshold Group Keys	✗	✗	✓	✗	✓
RBAC Authorization	✗	✗	✗	✗	✓
Tamper-Proof Logging	✗	✗	✗	✗	✓

F. Testing Results

We had six tests, which were unit, integration, system, performance, security, and usability.

* Each crypto primitive was tested individually, along with edge vectors in the interpolation of Shamir.

* Test one The tests used in integration checked that the keys of session were produced correctly using auth-to-msg flow.

* Nonces defeated security tests by demonstrating nonces can be resisted, and the log can be resisted by preventing tampering by checking with hash chains.

* End-to-end test on 500 simulated users registered no failures or corruption of data.

V. Conclusion and Future Work

We created one Python/Flask application, which combines ECC anonymous log-in, ECDH keys, AES-256-GCM, RBAC, Shamir group keys, hash-chain logs and encrypted cloud storage. It is stratified but slim, and it encompasses all the essential security characteristics that are missing in other articles.

The lab results indicate crypto ops not longer than sub-milliseconds (except PBKDF2), and therefore it is indeed possible to do real-time P2P chatting. The comparison table proves that we are the only model that has made it all come together. The test suite supports the idea that all is fine and we are not subject to replay and tampering.

Future outlook of the next version:

1. Outfit post-quantum KEMs such as CRYSTALS-Kyber to become quantum resistant.

2. Create a multi-cloud layer, which mirrors the data safely between providers, to be more accessible.

3. Connecting on biometrics as an addition in MFA factors greater identity binding.

4. Bek Write replacement of hash -chain audit log with a distributed ledger of tamper-proof accountability when used in hostile environment.

5. Expand to attribute-based access control to more fine context-sensitive policies.

Such upgrades would drive the AKA framework to the future-proof platform of secured team collaboration.

References

[1] W. Stallings, Cryptography and Network security: Principles and Practice, 8th ed. Pearson Education, 2020.

[2] J. Katz and Y. Lindell, Introduction to Modern Cryptography, 3 rd ed., CRC Press, 2020.

[3] F. Zhang, Y. Chen, and K. Yang, “Privacy-preserving authentication of cloud-based applications based on ECC, Journal of Computer Security, vol. 27, no. 4, p. 451 473, 2019.

[4] W. Diffie and M. -E. Challenge Hellman, new directions in cryptography, IEEE Trans. Inf. Theory, vol.22, no.6, p. 644 -654, 1976.

[5] A. Shamir, How to share a secret, Commun. ACM, vol. 22, no. 11, pp. 612-613, 1979.

[6] V. S. Miller, Use of elliptic curves in cryptography, in Proc.CRYPTO'85, LNCS 218, Springer, 1986, pp 417-426.

[7] N. Koblitz, Elliptic curve cryptosystems Math. Comput., vol. 48, no. 177, pp. 203-209, 1987.

[8] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin, Secure distributed key generation to discrete-log based cryptosystems, J. Cryptology, vol. 20, no. 1, pp. 5183, 2007.

[9] D. A. McGrew and J.Viega, The security and performance of the Galois/Counter Mode (GCM) of operation, in Proc.INDOCRYPT 2004, LNCS3348, Springer, 2004, pp.343-355.

[10] Recommendation for A Block Cipher Modes of Operation: Galois/Counter Mode (GCM), NIST SP 800-38D, 2007.

[11] S. S. Al-Riyami and K.G. Paterson, Certificateless public key cryptography, in Proc.ASIACRYPT 2003, LNCS2894Springer, 2003, pp. 452-473.

[12] R.S. Sandhu, E.J.Coyne, H.L.Feinstein and C.E. Youman, Role-based access control models, IEEE Computer, 29: 2, pp. 38-47 1996.

[13]B. Schneier and J. Kelsey, "Secure audit logs to support computer forensics," ACM Trans. Inf. Syst. Secur., vol. 2, no. 2, pp. 159 -176, 1999.

[14] Authors [14] H. Krawczyk, M. Bellare and R. Canetti, HMAC: Keyed -hashing of message authentication, IETF RFC 2104, 1997.

[15] J. Liu and P. Ning, "Efficient and secure key agreement protocols with distributed system in the IEEE Trans. Parallel Distrib. Syst., vol. 29, no. 6, pp. 1312-1325, 2018.

[16] Menezes, P. van Oorschot and S.Vanstone, Handbook of Applied Cryptography. CRC Press, 1996.

[17] C. Paar and J. Pelzl, Understanding Cryptography, 2nd ed. Springer, 2010.

[18] S. Subashini and V. Kavitha, Survey on security issues in service delivery models of cloud computing, J. Netw. Comput. Appl., vol. 34, no. 1, pp. 1-11, 2011.