Interdisciplinary Journal of Computing & AI

IJCAI
Interdisciplinary Journal of Computing & AI

IJCAI

RESEARCH

Privex: A Privacy-Preserving Online Meeting System Using Hybrid Cryptography

Tiyasa Paul[1*], Rohan Samanta[2], Pratik Saha[3], Partha Pratim Chaulia[4], Sudip Kumar Palit[5]

[1,2,3,4,5]*Department of Computer Science and Engineering (Artificial Intelligence & Machine Learning), Institute of Engineering and Management, University of Engineering and Management, Kolkata, West Bengal, India*
[5]*Cybersecurity Centre of Excellence, Institute of Engineering and Management, University of Engineering and Management, Kolkata, West Bengal, India*

Email: [1]shreetiyasa2003@gmail.com; [2]rohansamanta56@gmail.com; [3]pratiksaha964@gmail.com ; [4]prarthapratimchaulia@gmail.com; [5]sudipkumar.palit@uem.edu.in

ARTICLE INFO.

Keywords: End-to-End Encryption, Authentication, Non-Repudiation, Multi-Session Management, Co-Host Delegation, Message Automation

*Corresponding author email address: shreetiyasa2003@gmail.com

Manuscript received: 21 January 2026/Accepted April 10 2026
Published online: 17 April 2026
https://doi.org/10.5281/zenodo.19627442

Cite as: T. Paul, R. Samanta, P. Saha, P. P. Chaulia and S. K. Palit, "Privex: A Privacy-Preserving Online Meeting System Using Hybrid Cryptography", Interdisciplinary Journal of Computing & AI (IJCAI), vol. 1, no. 1, Aspiration Publishing Trust (DARPAN ID: WB/2025/0887371), p. 122-161, Apr. 2026, doi: 10.5281/zenodo.19627442.

ABSTRACT

In this paper, a behaviour-sensitive session orchestration architecture over secure communication and the rationale behind it will be proposed and discussed, which is a variant of mixed public-key cryptography that aims at offering more manageable, secret, and authenticity in real-time messaging infrastructure. The architecture incorporates the use of RSA and Diffie Hellman in ensuring the setup of user sessions and relies on AES to encrypt instantaneous messages very quickly. Multi-fernet algorithm supports the distribution of session keys and ensures that the key is recovered by the authorized parties only. There are also digital signatures based on RSA that facilitate authentication and non-repudiation but strong hash algorithms are applied to ensure the privacy of user passwords. The system has collaborative features along with cryptographic protection, including co-host delegation, active user blocking, permissions and multi-session to support multi-user logins. Also, the design

enables the dynamically updating of the authentication credentials without disruption of the existing communication. The suggested hybrid configuration has a higher level of defense against cryptographic attacks and unauthorized access than single algorithm implementations, which is why it is a more robust and secure solution. In conclusion, the provided solution contributes to the fact that the process of communication in the environment of multi-users becomes much safer, convenient, and manageable.

1 Introduction

Nowadays, as digital messaging is being delivered throughout the world, securing information being transferred over unstable, publicly available networks is a challenge in security studies. The methods such as RSA are useful in upholding privacy, accuracy and verification by use of encryption and signatures, but they are not always flexible. This algorithm helps to prevent the service disruption as well. In the meantime, the Diffie-Hellman method allows two parties to establish shared keys in a secure manner -even before prior coordination. Nevertheless, traditional methods are mostly ineffective in rapidly evolving circumstances when it is necessary to involve a group of users who need more intense control. The approach to public key enhances security and convenience by connecting RSA and Diffie-Hellman characteristics. Verified identities are used to create secure channels of chat with the help of hashed login information. Origin evidence is stored through digital marks through RSA. Even more, added functionalities were used to meet the real teamwork needs. Several sessions are supported and the host rights can change to co-host during a meeting. System alerts can be automated where limited to a constraint where password updates are still secured where it remains in active communication. The result of such upgrades is an increased flexibility throughout common cyberspace. The method employs concepts of various encryption systems to address vulnerabilities in individual systems, but enhance defence to common attacks. The testing was carried out in modelled environments in order to check on the safety measures as well as speed with varied hacking efforts. The following sections will discuss set up design, obstacles during the construction, and trial results with a combination of examples. It has been found that the model enhances teamwork tools over the internet without slowing down or complicating their use. It combines a variety of simultaneous sessions, shared hosting privileges, auto-messages can be disabled, denying any action is not possible and lets the user change their password safely on the same idle session. In contrast to the earlier models, which relied on one technique, this strategy can stand threats more efficiently, but change to various scenarios more readily.

2 Literature Survey

Considerable effort has gone towards developing both secure communication systems and cryptographic frameworks that enable the secure transfer of digital data between parties over a public channel. The current literature is able to be grouped systematically within five main research trajectories: (i) Secure communication and End-To-End Encryption (E2EE) (ii) Cryptographic Key Management and Exchange Protocols (iii) Integrity and Authentication of messages; (iv) Comparative Analysis of Hybrid Cryptographic Architectures; and (v) 5 Domain-Specific Security Architectures. A summary comparison of the different approaches on the basis of their security characteristics is included in Table 1.

2.1 Secure Communication and End-to-end Encryption

Numerous publications have been created to reduce weaknesses in validation using more advanced (and hybrid) encryption techniques and methods. Asha Jerlin *et al.* [1] have proposed an architecture for secure messaging using both RSA and AES (advanced encryption standards) to ensure data's confidentiality and the integrity of the message. Likewise, Almaiah *et al.* [2] developed a hybrid method combining elliptic curve cryptography (ECC) and Hill cipher-type systems to increase the security of mobile data transmissions from 'eavesdropping' or 'interception' threats. Likewise, Sivakumar et al. have developed a safety system for cloud-based communications using AES-RSA Hybrid Encryption combined with QKD (quantum key distribution) to ensure a 'future-safe' key. In addition, Fouzar *et al.* [3] have developed an ECC-based Hybrid Algorithm that is optimised for secure video streaming, which uses dynamic key scheduling in order to encrypt multiple video segments individually and to prevent unauthorised access to these segments. All of these studies demonstrate that hybrid cryptographic primitives successfully provide communication channels with robust protection from cyber-attacks.

2.2 Cryptographic Key Management and Exchange Protocols

The reliability of cryptographic systems highly relies upon good mechanisms for generating and distributing keys. To overcome the difficulties involved with transferring keys securely, Lein Harn *et al.* [4] created a decentralized way to establish a group key using mechanisms based on Diffie-Hellman functions and secret sharing instead of a central location (KGC) to generate keys. For cloud computing, Ahmad *et al.* [5] created an HCA-KMS (hybrid ECC and AES Key Management System) to assist with the management of cryptographic materials throughout their lifecycle in a virtualised environment. In addition, Sivakumar *et al.* [6] suggested the introduction of quantum key distribution (QKD) to strengthen the security of the key exchange process from quantum-based attacks. All of the methods discussed

here indicate that scalable and robust Key Management Systems (KMS) are essential to maintain the integrity of existing communication systems.

2.3 Integrity and Authentication of Messages

The security of today's communications infrastructures is fundamentally reliant upon two critical components: entity authentication and data integrity. To meet these needs, Asha Jerlin *et al.* [7] provide an integrated approach using the Diffie-Hellman protocol for a secure key exchange and session establishment, along with the Elliptic Curve Digital Signature Algorithm (ECDSA) and HMAC-SHA512 for message authentication and data integrity check upon receipt. Ahmad *et al.* [5] have also proposed a hybrid architecture to protect against impersonation attacks in the verification of entity-to-entity transactions that incorporate layered authentication in the cloud computing environments. Collectively, these approaches promote non-repudiation and also guarantee the integrity of data while it is being transmitted, forming a verified trust relationship between participants involved in communication.

2.4 Comparative Analysis of Hybrid Cryptographic Architectures

The synergy of symmetric and asymmetric cryptographic primitives has received much attention through extensive research efforts to optimize security and performance in secure online transactions and communication protocols. In their work analyzing hybrid cryptographic frameworks by Bhatele *et al.* [8], a detailed taxonomy was developed. The authors describe the basic premise of the hybrid methodology as follows: the symmetric algorithm provides computationally efficient processing of large quantities of data with high throughput; the asymmetric algorithm provides an additional layer of security for digitally signed messages and other types of electronic transactions by securely distributing robust keys that can be used to encrypt and decrypt data. By incorporating both methodologies into a single framework, hybrid systems are capable of providing higher security levels and greater data confidentiality, while at the same time maintaining low levels of latency and/or overhead within existing modern telecommunications networks.

2.5 Domain-Specific Security Architectures

Recent studies have focused on adapting cryptographic primitives to solve the unique constraints created in heterogeneous network types. An example of this would be Almaiah *et al.* [2], who developed a hybrid encryption scheme to make it more difficult for hackers to breach mobile communication protocol weaknesses due to wireless data transmission. Also, Fouzar *et al.* [3] implemented hybrid cryptographic architectures in multimedia streaming systems to provide the greatest data confidentiality and the lowest possible latency for real-time traffic. Both implementations show that there are

scalable cryptographic frameworks that may be customized to fit different application layer security demands.

2.6 Research Gap

When looking through the reviewed literature, we will see that there are many individual security solutions available such as encryption schemes, key management schemes, authentication schemes, or hybrid cryptographic schemes. Most of these solutions do not type in these components together to create one successful secure communication platform but tend to separate the elements out creating an unclear picture of what makes up a secure communication platform. The specified constraint of creating a solution with end-to-end encryption, non-repudiation, dynamic access control and secure session management added to the urgent need for the Privex system to exist. Table 1 represents the comparison of cryptographic techniques based on security and performance parameters is presented.

3 Proposed Approach

To ensure the safety, privateness, and reliability of communication in the app, a unique message and session system was created. Since it conforms to the basic requirements of security like secrecy, accuracy and accessibility, it is also useful in demonstrating the action taken in a court of law. Using such a structure, people can post information safely, authenticate themselves with whom they are communicating, can revoke permissions, block intrusions, altered posts, and bogus rejections at any time to come. Here a hybrid encryption strategy is adopted. To provide better key management and encryption of messages, the system uses multi-Fernet, which is a symmetric encryption utility that is offered by the cryptography package of Python. Each interviewee provides its own AES key, with which the multi-Fernet framework manages and merges to create a single master key. All session messages are then encrypted using this master key. This key cannot be accessed by anyone but an authorized user.

Privex uses a Server-Assisted End-to-End Encryption trust model, in which session messages are fully AES-encrypted at the endpoint of the sender and never leave the device, so the server is implemented as a ciphertext relay, and does not understand any content of the session messages at all - even a completely compromised server cannot read session messages. The session AES key is also secured by Multi-Fernet construct in which all the authorized participant key-shares are necessary to rebuild it and hence no one breach is possible on the server-side. The private keys of the RSA are stored in encrypted format only on the server, and they are encrypted over AES-128-CBC using a key derived with the personal password of the user via scrypt - a memory-hard algorithm that is difficult to brute force. The password used by the user is never stored or sent to the server in decipherable form and this

ensures that the server is the holder of the locked box but does not possess the key to unlock it.

Table 1 Comparison of cryptographic techniques based on security and performance parameters

Paper / Approach	Encryption Type	Security Features	Runtime Performance	Time Complexity
Ahmad et al. [5]	Hybrid (ECC + AES)	Confidentiality, Integrity, Key Management, Authentication, Secure Cloud Storage	Very High (Optimized ECC Keys)	O(log n)
Almaiah et al. [2]	Hybrid (ECC + Symmetric Hill)	Confidentiality, Authentication, Keyless Exchange, Secure ASCII Mapping	Moderate	$O(n^2)$
Jerlin et al. [1]	Hybrid Symmetric-Asymmetric	Confidentiality, Integrity, Authentication using AES + RSA, JWT for Login	High (AES-based)	$O(n^2)$
Lein Harn et al. [4]	Hybrid Diffie-Hellman	Key Authentication, Key Independence, Confidentiality	High	O(n)
Sivakumar et al. [6]	Hybrid + Quantum Key Distribution	Confidentiality, Integrity, Secure Key Sharing, Cloud Data Security	High (Cloud Computation Overhead)	$O(n^3)$
Bhatele et al. [8]	Hybrid (Symmetric + Asymmetric)	Integrity, Confidentiality, Authentication	Moderate	$O(n^2)$
Fouzar et al. [3]	ECC+Multi-Key Hybrid	Dynamic Keying, Multi-Layer Encryption, Anti-Piracy	High (Android Implementation)	O(n log n)

This hosted-key architecture is not a security compromise, but is intentional. Attributes that are core to the contribution dynamic cryptographic access revocation, co-host delegation, live key rotation and session-safe password updates need the server to actively participate in the key management

coordination, which is architecturally impossible with a pure zero-trust, device-local-key model. Privéx cryptographic architecture is specifically designed in a manner that all the sensitive operations message confidentiality, session key protection, and private key security are specified by mathematics and user-controlled credentials only, such that the system security assurances are universally true irrespective of the role played by the server in the communication process.

Decryption is carried out at the local level, i.e. end-users decrypt the data using renewed keys. This method guarantees privacy of information at every transfer level. When a user enters, exits or is removed out of a certain session the main key is updated dynamically, i.e., old access is no longer possible, but protection to new access is not compromised. The security is sensitive to the transforming conditions that are automatically rekeyed - continuity with exposure. The use of two-level RSA encryption may be appointed to offer non-repudiation - not repetition-based but a structural-check. Every user is given a single RSA key pair which is composed of a portion that is private and the other one is public. Privacy layer is ciphered with a password-based cipher and is stored in the server; others on the other side can however access the public one when communicating. When one logs in, he or she is decrypted using his or her passphrase of the user and the secret key is loaded into a temporary run time memory. This stored personal information is manifested in the tagging of the active and outgoing messages with a digital stamp when they are in use. When receivers get them, they validate the mark with a counterpart that is obtainable in the open of the broadcaster. Such validation ensures that every object dispatched is always closely tied to its place of origin - eliminating future arguments of non-involvement. This contract supports predictability of law suits in which evidence is a matter of count.

A live blocking tool was created in a manner that enables hosts to have control of hosts when they are in open meetings. The method relies on the Diffie-Hellman to come up with temporary secret keys between the host and the attendees. When one gets blocked, a number of things happens at the same time; a system issues a blocked flag in the database, issues new key sets to non-blocked people, and pulls the primary session key. Thanks to this fact, the excluded person will be unable to use his/her old key any more - the information on the access to the cache is automatically vacated. Any attempt by the blocked user to rejoin is rejected instantly as at the encryption level there are invalid. In addition to providing a superior privacy and control, the design enables rapid updates and stable sessions in case of a group work. The security is improved through using HTTPS connections, encrypted login codes, and HMAC authentication of the trust of the sessions and managing the active states with Flask and Flask-Socket IO. On a cumulative basis, the methodology creates strong, scalable security, which satisfies the current regulations of safety and which adjusts to the evolving requirements in the common digital landscapes with ease.

Figure 1 presents the flowchart of the proposed approach, illustrating the backend meeting initiation process, dynamic blocking mechanism, and secure key establishment using RSA and Diffie–Hellman. It shows how clients are authenticated, access is controlled through block and key verification, and session security is maintained by regenerating shared keys whenever user participation changes.

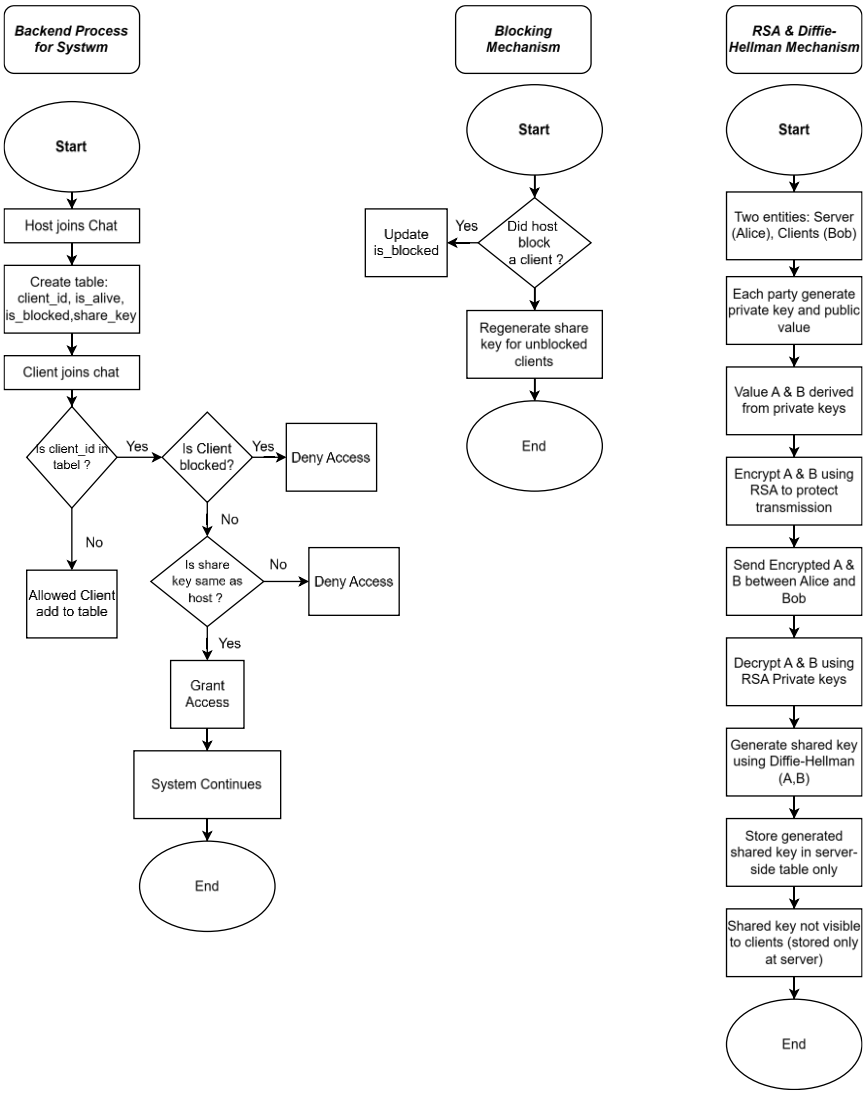

Fig. 1 Flowchart of Proposed Approach

3.1 Implementation of Proposed Approach

The proposed secure communication architecture is built around a modular, protocol-based Python architecture, which is heavily customized on the Flask web framework and Flask-SocketIO to handle real-time sessions. Flask session-based authentication is utilized to carry out user authentication and session management with user credential safely processed using hashlib module with hash-256 hash and stored in SQLite database. The communication between the client and server is fully carried out by means of HTTPS (TLS), which offers safe encryption of the transport-layer. Live meetings are dynamically controlled by the backend which manages active state of sessions, user roles (host/co-host/participant) and access permissions, making it possible to fine-tune live meetings.

The hybrid symmetric-key made on the basis of Secret Sharing and cryptography Python module are used as an end-to-end encryption technique. The generation of individual AES keys per participant with the assistance of the Fernet scheme is used in each meeting session. These keys along with purpose are combined using a Multi-Fernet construct whose purpose is a secret-sharing layer disclosing a master encryption key with no corresponding plaintext. The master key is used to encrypt AES key with all the message content which is session specific. The encryption and decryption are performed at the client-end point alone hence, actual end-to-end confidentiality. Every time a participant joins, leaves or gets stuck, the system triggers an automatic key rotation scheme which reconstructs participant keys and re-encrypts the session key in such a way as to provide both forward and back secrecy without stopping the running session.

Non-repudiation and access control can be enforced by using public-key cryptography and key management. The creation of key pairs of RSA via Crypto. Public Key, RSA, as well as the encryption of the private keys with the help of password-based key derivation (scrypt with AES-128-CBC) and subsequent secure storage, is done. Messages sent are digital- signature signed with the RSA private key of the message sender and authenticated at the receiver side by the RSA public key which in turn is authenticated by the SHA-256 hash function with the PKCS1 authentication signature. Traceability All the signature and verification process is logged to append-only audit files. This is done through a dynamic blocking algorithm that runs on Diffie-Hellman key exchange protocol whereby shared secrets are re-computed each time a user is blocked. This invalidates the old session keys and tokens accumulated before and invalidates them all at once thus preventing unauthorized re-entry or decryption of messages. Co-host delegation further enhances availability because it supports a tying together of session control delegation so that the continuity and security of collaboration can be maintained throughout the meeting life cycle.

3.2 Algorithms

3.2.1 End-to-End Encryption Algorithm

The encryption plan proposed will protect chat data between the participants in a mixed AES key-based set-up. In a Multi-Fernet model where each user is given a unique key of AES that is re-generated each session, keys are concatenated so as only the authorized users will be able to recombine to get the original decryption key. This primary key, in its turn, freezes a new AES session key, whereas the same AES session key codes all messages sent during the meeting. This is called end-to-end encryption because the AES key that is used to encrypt a message is not known to the servers and middle systems. The decryption rights are only granted to the verified users on both sides, therefore, the only people that can unlock the content. Information is encrypted as it passes and it is only relayed by the node without access tools. Thus, once the attackers are getting access to the internal systems, logs or in personnel assistance, secrecy is maintained. The layer is the most significant one in case of virtual meetings when personal talks, financial papers or personal communication should better remain beyond the reach of the unwary persons. The implication of this is that keys are re-used when an individual joins or leaves a session such that the previous participants cannot decrypt the following messages (forward secrecy), and the new ones cannot read the previous messages (backward secrecy). Whenever such an update is done, new keys are invented automatically and swapped on ensuring that the level of protection is high even when the groups are swapped frequently. No security is lost that slows down performance. The core end-to-end encrypted message in the Privex system, which is used to safeguard all the communications that occur during meetings is described in the Pseudocode 1. Through it, the messages are encrypted and decrypted by authorized members only and the server is used as a relay and has no access to plaintext messages. The algorithm uses dynamically regenerating cryptographic keys when the membership of the participants varies thus maintaining the confidentiality as well as forward and backward secrecy all through the meeting lifecycle. This same end-to-end encryption and key regeneration process is illustrated in Figure 2.

Pseudocode 1 End-to-End Encryption

```
Initialize E2E Encryption Module
FUNCTION E2E.initialize()
 Generate AES keys for all users in a meeting
FUNCTION create_AES_keys_for_users(users, meetingId)
FOR each user IN users DO
key ← Generate random Fernet key
STORE key in MeetingKeysDb(user, meetingId)
END FOR
```

```
9.  End function
10. FUNCTION get_master_key(meetingId)
11. keys_list ← FETCH all keys from MeetingKeysDb USING
    meetingId
12. master_key ← MultiFernet(keys_list)
13. RETURN master_key
14. End function.
15.  FUNCTION set_Aes_key(meetingId)
16.  plain_key ← Generate new Fernet key
17.  master_key ← get_master_key(meetingId)
18.  encrypted_key ← Encrypt(plain_key, master_key)
19.  STORE encrypted_key in MeetingAES(meetingId)
20. End function
21.  FUNCTION format_key(meetingId, users)
22.  encrypted_key ← FETCH encrypted key FROM MeetingAES
23.  master_key ← get_master_key(meetingId)
24.  decrypted_key ← Decrypt(encrypted_key, master_key)
25.  CALL rotate_keys(meetingId, users)
26.  new_master_key ← get_master_key(meetingId)
27.  re_encrypted_key ← Encrypt(decrypted_key, new_master_key)
28.  UPDATE MeetingAES with re_encrypted_key
29. End function
30.  FUNCTION encrypt(msg, meetingId)
31.  encrypted_key ← FETCH encrypted key FROM MeetingAES
32.  master_key ← get_master_key(meetingId)
33.  actual_key ← Decrypt(encrypted_key, master_key)
34.  ciphertext ← Encrypt(msg, actual_key)
35.  RETURN ciphertext
36. End function
37.  FUNCTION decrypt(token, meetingId)
38.  encrypted_key ← FETCH encrypted key FROM MeetingAES
39.  master_key ← get_master_key(meetingId)
40.  actual_key ← Decrypt(encrypted_key, master_key)
41.  plaintext ← Decrypt(token, actual_key)
42.  RETURN plaintext
43. End function
44.  FUNCTION rotate_keys(meetingId, users)
45.  DELETE existing keys FROM MeetingKeysDb USING meetingId
46.  CALL create_AES_keys_for_users(users, meetingId)
47. End function.
48. End.
```

3.2.2 Non-Repudiation Using RSA Digital Signatures Algorithm

This system may support non-repudiation property whereby each message sent during a given session is linked to a real source in a secure manner where the parties concerned will not be able to denounce the involvement.

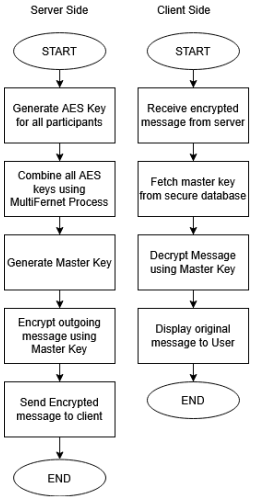

Fig. 2 Flowchart of End-to-End Encryption

The lack of ability to complain later that a message was not written or sent by the user is that since messages contain the transmissions of the digital signatures that are signed with a private key, the association between an individual and a message can resist verification. The latter plays a crucial role in the secure messaging arrangements, particularly in the team work where the sense of responsibility is observed, as the latter grants the monitoring of the actions, as well as the reliability of the latter. This is what has been termed as non-repudiation since the process could not be repudiated by the cryptographic evidence which is directly linked to the individual. To achieve this, users create own RSA key pairs upon registration.

In storage, the private key is encrypted by a password mechanism - that is, scrypt with AES-128-CBC - which is only decryptable by the owner at a later stage. Quite to the contrary, the public key is shared by free means and the digital signatures can be verified by other people. In order to send passwords, the source cannot only compute a SHA-256 digest of the original text, but he or she signs with his or her own secret key and he or she adheres to the PKCS1 v1.5 protocol. Together with this cryptography evidence, additional information about who sent it, the time, and the session identifier are recalled as one and creates an indelible trace that would make it difficult to alter. When the system receives a message, it fetches the associated public key and authenticates the signature by recomputing the hash of the message. Where the check is a match, it is checked as verified, otherwise, not verified. Every effort irrespective of failure or success is kept in a different record to be traced. The mechanism will also ensure that the messages are not tampered with, identities of the sender can be determined, accountability upheld, and in the process, any message relayed by any source can be traced to its source without any doubt.

The Privex system is made to facilitate the accountability and non-repudiation of a secure communication. Pseudocode 2 describes the steps of this secure communication. It does it by signing every message digitally with the RSA privative key of the sender and validating the signature with the public key. This will ensure that all messages are easily traced into their source, thus eliminating the denial of the message being authored and enhancing trust among the involved parties. Also, the Pseudocode 2 enables auditability when meetings are held collaboratively. The same image (Figure 3) shows the message signing and verification process, which is a visual representation of the activity of the pseudocode 2.

Pseudocode 2 Non-Repudiation USING RSA Digital Signature

1. **START**
2. **CHECK** if password IS provided
3. **IF** password = NULL THEN
4. **RAISE** "Password required for key encryption"
5. **END IF**
6. **GENERATE** RSA key pair (2048 bits)
7. **ENCRYPT** private key using scrypt + AES-128-CBC with password
8. **SAVE** encrypted private key AS <username>_private.pem
9. **SAVE** public key AS <username>_public.pem
10. **LOAD** private key using password
11. **COMPUTE** SHA-256 hash of the input message
12. **SIGN** hash USING private key (PKCS#1 v1.5)
13. **CONVERT** signature **TO** hexadecimal
14. **LOG** (sessionId, username, signature, timestamp) **INTO** chat_logs.json
15. **RETURN** signature_hex
16. **LOAD** public key of sender
17. **COMPUTE** SHA-256 hash of received message
18. **CONVERT** received signature_hex TO bytes
19. **VERIFY** hash USING public key (PKCS#1 v1.5)
20. **IF** verification SUCCESS THEN
21. status ← "verified"
22. **ELSE**
23. status ← "not verified"
24. **END IF**
25. **LOG** (sessionId, username, status, timestamp) **INTO** chat_logs_verified.json
26. **RETURN** verification status
27. **END**

3.2.3 Dynamic Blocking System Algorithm

The dynamic block system facilitates access control in broadcasting, which is safe in real time. Dynamic because it can put a user instantly offline and create new encryption keys in real time - and leave other individuals offline. This ensures that the message is secret and trustful because the members who are not part of the group will never be aware of the future messages and the risk of an expelled member coming back is also not present due to the assistance of the older data. In its absence, this operation can be a security vulnerability, with breaches in reused tokens or the existing exchanges will be visible. The system works in accordance with the RSA to administer identity-related protection as well as the Diffie-Hellman (DH), which produces many symmetric keys of varying lengths. The first step is to create RSA sets by the session leader of that particular session; the remainder of the participants are then given the public part of the set.

In order to derive secure symmetric key, DH values (P and G) are generated - this allows every participant involved be it an organizer or an attendee to develop his or her own DH key, both the private and the public key. Both sides are swapped with public keys to come up with a shared secret. It is built on that shared output to build the encryption keys that are necessary on the session. In the case when the host has been banned, the site instantly removes the shared secrets according to the DH and erases all the information published previously, and breaks the current connections. When this happens, new DH values are obtained - which immediately renews the key of all the already-running members. This process and new secret pairs are created, and the material used in the old encryption will not be of any use to a deserted person. DH also offers forward secrecy to have an added bulk of security where any keys stored or sent will not be helpful when they are removed - making the connections of the connections safe and steady throughout the entire session.

To prevent the user privileges of a blocked user, the pseudocode 3 is created to implement real time access control and security of a session by revoking the privileges of the user immediately. On revocation, the algorithm causes the regeneration of Diffie-Hellman shared keys and voids all the previously utilized encryption content. This makes sure that participants that are removed cannot decrypt any further communication or resuming the session with the help of a cache. Pseudocode 3 makes forward secrecy strong and tightly couples the access control with the cryptographic protection during the entire lifecycle of the meeting by dynamically rekeying the session. This revocation and rekeying process is also shown in (Figure 4), and it is a graphical depiction of how the pseudocode 3 should work.

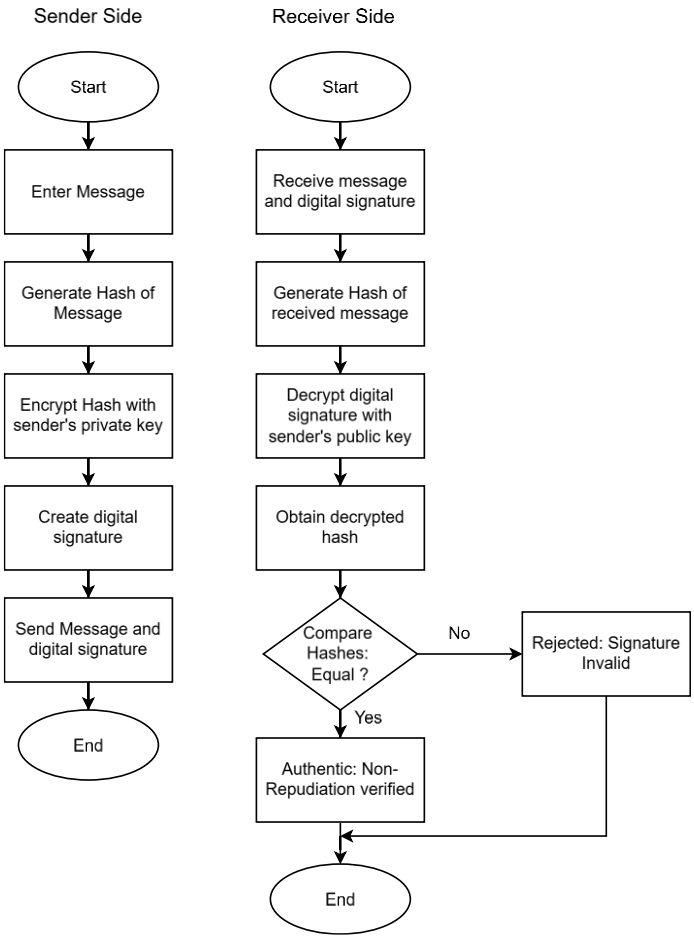

Fig. 3 Flowchart of Non-Repudiation

Pseudocode 3 Dynamic Blocking System

```
Initialize RSA Key Infrastructure
FUNCTION generate_meeting_keys_RSA(meetingID)
CREATE a pair of RSA public and private keys (2048 bits)
STORE keys as <meetingID>_public_key.pem and
<meetingID>_private_key.pem
END FUNCTION
FUNCTION RSA_Encrypt(public_key, message)
IMPORT public_key
ENCRYPT message using PKCS1_OAEP
RETURN base64 encoded ciphertext
END FUNCTION
```

```
FUNCTION RSA_Decrypt(private_key, ciphertext)
IMPORT private_key
DECRYPT ciphertext using PKCS1_OAEP
RETURN plaintext message
END FUNCTION
FUNCTION generate_dh_parameters(bits)
VALIDATE bits is multiple of 128 and > 512
q ← Generate large strong prime number of 'bits' length
g ← 2
RETURN (q, g)
END FUNCTION
CLASS DiffieHellman(p, g)
GENERATE private_key ← Random(2, p−1)
COMPUTE public_key ← g^private_key mod p
FUNCTION compute_shared_secret(received_public_key)
RETURN received_public_key^private_key mod p
END FUNCTION
END CLASS
FUNCTION generate_new_dh_key(username, meetingID, P, G)
host_dh ← DiffieHellman(P, G)
server_dh ← DiffieHellman(P, G)
host_secret ← host_dh.compute_shared_secret(server_dh.public_key)
server_secret ←
server_dh.compute_shared_secret(host_dh.public_key)
RETURN (host_secret, server_secret)
END FUNCTION
FUNCTION dynamic_block_user(blocked_user, meetingID)
MARK blocked_user.is_blocked ← True
FETCH updated active users list
//y Regenerate Diffie−Hellman shared keys for all remaining users
P, G ← generate_dh_parameters(1024)
FOR each user IN active_users DO
(user_key, server_key) ← generate_new_dh_key(user, meetingID, P, G)
UPDATE session_key(user, meetingID, user_key)
END FOR
DELETE blocked_user's old key pair FROM key database
NOTIFY all active users about session rekey event
END FUNCTION
IF host initiates blocking event THEN
CALL dynamic_block_user(blocked_user, meetingID)
REKEY all user sessions
END IF
End.
```

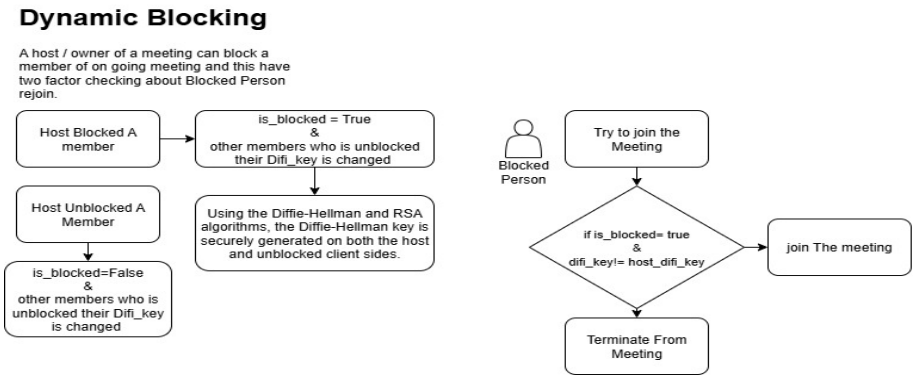

Fig. 4 Flowchart of Dynamic Blocking System

4 Resource & Technologies Used

4.1 Resources Used

The proposed system was developed and tested on the existing desktop and laptop systems that are capable of supporting the existing Python based web applications. It is supported on the Python 3.10 or later, and this choice is made as it is stable, has a comprehensive standard library, and may be combined with advanced cryptographic and networking libraries. Flask web framework is used as the back end that enables the use of secure sessions and RESTful development of services and Flask-socketIO is added to give real-time and two-way communications required during the live meeting, delivery of instant messages, dynamic blocking and co-host role administration. SQLite is a relational database that is based on user credentials, hashed passwords, session-data, encryption metadata, and access-control states since it is lightweight with low overhead and predictable transactional properties. Gunicorn is used to control deployment, where it is a WSGI-based server that is production-ready in order to have the requests processed effectively and scale to a number of users.

Cryptographic operations are done using a set of pre-established Python libraries. Cryptography library is implemented on AES-based symmetric encryption and Fernet and Multi-Fernet key management and safe end-to-end encryption processes. Cryptographic uses of the public key cryptography such as the RSA key generation, non-repudiation digital signature, and dynamic session rekeying using Diffie Hellman key exchange are done using the PyCryptodome (Cryptodomex). The hash of user passwords is computed using the hashlib provided with the built-in hashlib module implementing the SHA-256 hash and without storing the plaintext credential, and eliminating the risk of offline attacks. All of these technologies constitute a powerful, stable, and scalable platform that would facilitate the hybrid

cryptography protocols, real-time session management, and efficient access control in the communication platform suggested.

4.2 Technologies Used

The suggested cryptographic and session management framework required integration of a number of secure communication protocols and cryptographic mechanisms to provide confidentiality, integrity, authentication and non-repudiation. The architecture of the system is based on the hybrid encryption approach which is a combination of symmetric and asymmetric encryption methods. Advanced Encryption Standard (AES) [9] is used to obtain end-to-end encryption with individual user keys being safely concatenated with the principles of Multi-Fernet [10] based encryption to obtain a safeguarded session key. This session key is dynamically regenerated every time people join, leave, or are blocked, in order to have forward and backward secrecy all the time. Communication with partying entities through secure means is enforced with Transport Layer Security (TLS) [11] which avoids eavesdropping and tampering in message transfer. Strong authentication and accountability require that password-based authentication be carried out with the help of the cryptographic hash functions so that sensitive credentials are not stored or transferred in plaintext. Non-repudiation is carried out using RSA-based digital signatures [12] whereby each message is signed cryptographically by the sender and is checked by recipients through public-key checking. Password-based encryption is applied to the protection of the private keys which are only activated temporarily on authenticated sessions. The integrity of the session itself is further enhanced with the help of the signed authentication tokens and the message authentication codes to avoid the attacks of replay and impersonation. The reduction of verification latency is achieved with the use of public-key distribution and caching mechanisms. Together, these protocols and mechanisms constitute a resilient, secure and scalable communications framework that can be adapted to dynamic session changes and provide and sustain strong cryptographic guarantees.

5 Security Analysis

The proposed hybrid public key system will integrate a number of crypto-tools such as AES, Multi-Fernet [10] a symmetric encryption RSA signatures, as well as Diffie-Hellman, to provide a high-quality, full cover protection. In this case, we examine the system security level; simultaneously, investigate the resistance of the system to the common attacks.

5.1 Confidentiality

Security is also based on End-to-End Encryption [13] which is powered by AES [9]; in this case, the session key remains hidden by a multi-Fernet [10] configuration using user-specific AES codes. Intruders cannot decrypt messages unless they have all the necessary parts as the primary AES code is only created when the proper parts are combined. When a user enters, exits, or removes, the primary key is rebuilt with new pieces - the previous communications remain secure as well as the future communications are secured against easy surveillance by a simple spying - using the secure channels, no pieces of the data are leaked when transferring, and because the connections are encrypted by default, the systems cannot easily access the communication.

5.2 Integrity

The integrity of messages is ensured at two levels:

- AES Encryption Integrity: Fernet encryption relies on the HMAC [14] to test the integrity of data, and thus any manipulations with encrypted data can be noticed instantly.
- RSA Digital Signatures: Each message is signed with a digital signature of the secret key of a sender (secured through scrypt + AES-128-CBC). When a person changes it, RSA public-key verification indicates the change.
- Messages are stored in write-once formats, which prevent the ability to alter or forge messages in history. In combination, these two features make it impossible to alter or forge a message.

5.3 Authentication

- The system has various verification procedures with one being undertaken after another.
- Password hashing with SHA-256 will imply that even in case the database is compromised, the credentials are not retrievable.
- RSA private keys which are password protected enable only authorized individuals to use their decryption data by using secure authentication procedures.
- Due to the fact that messages require a personal key - and a key is unlocked upon your signing in - the origin remains validated.

5.4 Non-Repudiation

The RSA protocol ensures that every message can be successfully authenticated to the sender.

- Messages are instead signed using a private key.
- Check records which include working - and not working - are maintained in tamper-proof log files.

- A user is not able to deny that he/she sent a message later - this satisfies traceability requirements in legal situations.

5.5 Availability

The system provides availability by:

- Fast AES coding is ideal in real time data transfer since it maintains the speed of data transfer and also keeps security high.
- Rapid DH key renewal without breaking of active connections.
- Co-hosts replace the absent host and the sessions continue to operate without interruption. This removes any one point of failure while keeping meetings running smoothly.

5.6 Attack Categories

5.6.1 Attacker Type 1: Network Attacker (Dolev-Yao Adversary)

Definition: A Network Attacker A_{net}possesses full control over the communication channel, capable of intercepting, replaying, or injecting packets at will. The adversary has no valid credentials (id_X, pw_X) or session keys$K_{session}$.

Threat Analysis: The system is secure against A_{net} if the probability of breaking the end-to-end encryption (E2EE) is negligible:

$$Pr[A_{net}(C) \rightarrow M] \leq \epsilon(\lambda)$$

Where C is the intercepted ciphertext $E(M, K_{session})$. Because the master key K_{master} is derived via a multi-Fernet construct requiring n individual user keys:

$$K_{master} = MultiFernet(k_1, k_2, \dots, k_n)$$

An attacker without these keys cannot derive the plaintext. Furthermore, replayed ciphertexts are invalidated via session re-keying:

$$K_{session}^{(i)} \neq K_{session}^{(i+1)}$$

Where $\$i$ denotes the session state. Thus, A_{net} cannot recover key material or read session data.

5.6.2 Attacker Type 2: Malicious Participants

Definition: A Malicious Participant A_{int} is a legitimately authenticated user within session i. They possess a valid RSA key pair (PK_{int}, SK_{int}) and a specific AES key-share (k_{int})

Threat Analysis: To impersonate another user U_j , the adversary must forge a digital signatureσ_j:

$$\sigma_j = Sign\left(SK_j, H(M)\right)$$

The security of the system relies on the hardness of the RSA factoring problem. The probability of A_{int} successfully forging σ_j is:

$$Pr\left[A_{int}\left(PK_j, M\right) \rightarrow \sigma_j\right] \leq \epsilon(\lambda)$$

Additionally, since the session master key K_{master} requires all authorized shares, partial possession by A_{int} yields zero information:

$$H(k_{int}) \approx Uniform$$

All actions are bound by digital signatures stored in append-only audit logs, ensuring accountability.

5.6.3 Attacker Type 3: Removed User (Forward/Backward Secrecy)

Definition: A Removed User A_{rem} is an entity whose status has been updated to $is\_blocked = True$. They retain legacy key material $K_{session}^{(i-1)}$ from previous participation.

Threat Analysis: The Privex system enforces Perfect Forward Secrecy (PFS) and Backward Secrecy through dynamic re-keying. Upon a membership change, new Diffie-Hellman (DH) parameters are generated:

$$SS^{(i)} = g^{ab} \backslash pmodp$$

Where $SS^{(i)}$ is mathematically independent of $SS^{(i-1)}$. The probability of A_{rem} using stale material to decrypt future messages $C^{(i)}$ is:

$$Pr\left[A_{rem}\left(K_{session}^{(i-1)}, C^{(i)}\right) \rightarrow M^{(i)}\right] \leq \epsilon(\lambda)$$

The $is\_blocked$ flag acts as a non-cryptographic barrier that terminates the protocol before cryptographic processing even begins.

5.6.4 Attacker Type 4: Compromised Server Attack

Definition: This model assumes the adversary A_{srv}has gained administrative access to the server's database and memory.

Threat Analysis: Privex utilizes a Server-Assisted E2EE model. Sensitive data is never stored in plaintext:

- Passwords: Stored as $H(pw\_X)$ using SHA-256.
- Private Keys: Encrypted as $E(SK_X, scrypt(pw_X))$
- Session Keys: Encrypted via multi-Fernet; the server only acts as a relay for ciphertext C.

Even with full database access, the probability of A_{srv} recovering a private key without the user's password is:

$$Pr[A_{srv}(Encrypted\ SK_X) \rightarrow SK_X] \leq \epsilon(\lambda)$$

Because scrypt is a memory-hard function, brute-force attempts are computationally impractical.

5.6.5 Attacker Type 5: Network Eavesdropping

Definition: All end-to-end session messages are encrypted through end-to-end AES encryption, which means that any information intercepted will not be readable. The use of secure key exchange and signature verification between the Client and Server will prevent the compromise or forgery of the session key used in order to maintain the confidentiality of the messages sent between both Client and Server, as well as maintaining the integrity of the messages sent between both Client and Server from any network adversaries.

Threat Analysis: Read confidential session messages sent in the session; collect ciphertext transmitted in the session; deploy analysis to determine communication patterns; attempt to reconstruct plaintext from the captured data within the network.

Protection Mechanisms that are Mapped to this Threat:

- All messages sent within the session are encrypted end-to-end using AES via the Fernet encryption scheme before leaving the sending device.
- The communication channel is being secured using TLS, to provide encrypted communication between the Client and the Server.
- The Server does not process plaintext messages. The only message the Server/Client can process is ciphertext.

Even if an eavesdropper captures all packets traveling within a network, the communication that they intercepted is still going to be unreadable ciphertext (unless the eavesdropper is in possession of the session key). An eavesdropper who captures the communication transmitted will not gain any valuable information due to not having possession of the session key, therefor the eavesdropper will not have any useful data regardless of the number of packets they have captured.

5.6.6 Attacker Type 6: Man-in-the-Middle Attack

Definition: A man-in-the-middle (MITM) attack is when an attacker secretly intercepts and may also modify communications between two parties. An attacker can place themselves in between the sender and the receiver to listen to, modify, or insert into a communications session.

Threat Analysis: DENY: The attacker can intercept session messages; modify messages during transport; insert malicious messages into sessions; impersonate legitimate participants during sessions.

Protection mechanisms for this Threat:

- TLS-secured transport ensures no unauthorized interception of messages and confirms the authenticity of the connection.
- Each message includes an RSA-2048 digital signature that the recipient can verify to confirm the identity of the message's author.
- Diffie-Hellman key verification securely negotiates shared keys between participants.

A successful MITM cannot modify, insert, or impersonate a message without detection, in that all modifications to any message will cause the signature verification or the TLS validation to fail, and the message will be rejected by the recipient.

5.6.7 Attacker Type 7: Replay Attack

Definition: This is an attack where an attacker manipulates or falsifies data transmitted after it has been legitimately sent by the sender, to gain unauthorized access to something, or disrupt a service.

Threat Analysis: Capture existing ciphertext and re-use this ciphertext to re-play previously valid function calls, or messages and disrupt the integrity of a session by injecting messages into the existing session that are stale.

Protection Mechanisms Mapped to This Threat:

- Session AES keys are automatically rotated when any participant joins, leaves or has been blocked.
- Fernet encryption includes HMAC-SHA256 for integrity verification of each message ciphertext.

When the session key changes, old ciphertext becomes invalid and cannot be decrypted until the key has been changed, and therefore the captured data will also be deemed cryptographically stale immediately after key rotation. All previously replayed messages will fail HMAC validation and as a result will be rejected prior to their decryption.

5.6.8 Attacker Type 8: Message Forgery Attack

Definition: An attacker tries to impersonate a legitimate participant by fabricating and transmitting messages from the impersonated party to trick counterparts into believing they are receiving authentic messages.

Threat Analysis: Any messages can be fabricated using another's identity; there is an ability to inject unauthorized content into the conversation; an ability exists to manipulate the flow of a conversation.

Protection Mechanisms Mapped to This Threat:

- The RSA-2048 digital signature must be present in every message when generated by the original sender's private key.
- The signature generated on the messages uses SHA-256 Hashing algorithm with the PKCS#1 v1.5 Verification Method.
- Verification of all message signatures must take place prior to any messages being accepted.

An attacker cannot create a valid signature if access to the secret/private key belonging to the actual sender has not been granted. Furthermore, any forged message will never pass verification and will be rejected immediately.

5.6.9 Attacker Type 9: Message Tampering Attack

Definition: Unauthorized alteration of a legitimate message during transmission constitutes modification in violation of the integrity requirement with the intent of avoiding detection.
Threat Analysis: Change content of message; corrupt encrypted data; change meaning of the message by altering transmitted ciphertext.

Protection Mechanisms (Mapped to this Threat):

- Every encrypted message (using Fernet encryption) will use HMAC-SHA256 for authentication.
- Modification to the ciphertext will invalidate HMAC verification.
- The application layer uses a RSA digital signature to independently verify message integrity.

Detection of any modifications to the ciphertext occurs pre-decrypt. All tampered messages fail to verify integrity and will therefore be rejected by the system.

5.6.10 Attacker Type 10: Brute-Force and Cryptographic Resilience

The system employs high-entropy keys to prevent exhaustive search attacks:

- AES-128: Provides a key space of 2^{128}.
- RSA-2048: Relies on the integer factorization complexity.

$$Factoring\ N = p \cdot q\ where\ |N| = 2048\ bits$$

Computational infeasibility is maintained as long as $\lambda \geq 128$ for symmetric and $\lambda \geq 2048$ for asymmetric components.

5.6.11 Attacker Type 11: Private Key Theft Attack

Description: An attack against a user's private cryptographic key in order to gain access to the decryption of encrypted messages, or to create a digital signature that impersonates the user.

Threat Analysis: Stealing private keys to create false signatures and impersonate legitimate users' identities to gain unauthorized authorization for a message.

Defensive Mechanisms to Counter the Risk:

- RSA private keys will be encrypted with AES-128 CBC.
- The key to the private key will be generated using the scrypt password-based key derivation function.
- User passwords will not be stored in the system.

Utilizing a password related to a private key will render the private key useless. Attempting to perform a brute-force search for a password on a private key will take many years, due to the design of the memory-hard nature of the scrypt algorithm.

5.6.12 Attacker Type 12: Forward/Backward Secrecy Attack

Definition: This attack scenario includes forward secrecy failure when an adversary is capable of successfully decrypting previously sent messages after obtaining a compromise on a current (or prior) key and the backwards secrecy failure, where an adversary is capable of decrypting future messages based upon having obtained a compromise on the currently used key.

Threat analysis: This scenario allows an adversary to be able to use the current key to decrypt prior (or historical) messages; and also allows an adversary to be able to access a conversation history that was established prior to joining, as well as decrypting future messages after having left the session.

Protection Mechanisms that map to this Threat:

- Every time there is a change of participant membership, Diffie-Hellman re-keying generates a new and unique session key.
- The rotate_keys() mechanism generates new keys when any user joins or leaves the session.
- There is no mathematical relationship between the old keys and the new keys.

Even if the current key is compromised, past messages are protected and cannot be decrypted by new users that did not participate in the session prior to joining.

5.6.13 Attacker Type 13: Repudiation Attack

Definition: Defined as attempting to disavow / deny that you have done (action) or sent (message) while exploiting the fact that there is no verifiable proof mechanism available (i.e., evidence of the action has not occurred).

The following items fall under this category of threat:

Denying authorship of message(s) or avoiding responsibility for message(s) sent; questioning where messages came from (origin).

The following protection mechanisms have been mapped to this category of threat:

- All messages are digitally signed using RSA encryption/signature
- All digitally signed messages are maintained in Audit Logging in an Append Only Format
- Each record in the Audit Log contains the session id, the user name, the signature of the user that created the message, and date/timestamp

Having the sender's Public Key for a given message, allows you to verify authorship. The signed Audit Log records of messages provide a permanent, verifiable proof of who created the message.

5.6.14 Attacker Type 14: SQL Injection Attack

Definition: SQL injection attacks occur when input validation is performed improperly and allow an individual to insert malicious SQL statements into a SQL command, resulting in potentially obtaining access to, modifying or deleting data within a database (or data source).

Threat Analysis: This threat can extract confidential data from a database, modify stored information already within the database and bypass the authentication mechanisms employed by the database.

Protection Mechanisms That Mitigate This Threat

- All SQL statements that modify the database employ the use of parameterized queries through SQLite (i.e. Bound Variables).
- All input to a SQL statement is verified against strict input validation rules prior to the statement being processed.
- We have utilized a static analysis tool (i.e. Bandit) to identify and eliminate unsafe SQL statements from our application.

Therefore, SQL injection vulnerabilities will be addressed at the implementation level. As a result, there are no unauthorized database queries that can be executed by attackers.

5.7 Secure Implementation Validation and Static Code Analysis

In advance of the implementation of this proposed system, we performed a thorough static code analysis to verify the security integrity of the software. We used the Bandit (v1.8.6) tool to perform a pre-deployment scan on all the application's modules and on all dependencies in the virtual environment on August 19, 2025, with the total number of lines of code being 1,099,003. Once we found vulnerabilities in the scan, we fixed them, and after fixing the security vulnerabilities, we scanned the codebase again to verify that we had fixed the identified vulnerabilities. The results of the findings, the actions we took to fix the vulnerabilities, and the results from the re-scanning for remediation are all contained in this section, which serves as evidence that the system has performed all required security assessments to be in compliance.

CWE is a list of common software and hardware vulnerabilities that provides standardized definitions of the types of vulnerabilities that can exist as a result of a flaw in the design or implementation of the software, or the configuration of the device. CWE is maintained by the MITRE Corporation and is used as the basis for identifying and categorizing potential weaknesses in a system so they can be methodically assessed and remediated in the security assessment and validation process.

This research employs CWE to develop a framework for assessing the proposed architecture's ability to withstand attacks exploiting known categories of Software security weaknesses (e.g., misuse of cryptography, unauthorized access to the key management system, or flaws in authentication). The mapping of potential weaknesses and attack scenarios against relevant CWE categories helps validate that the proposed system was designed securely and mitigates exposure to known types of Software Security vulnerabilities. Therefore, using the CWE mapping strengthens the Security Analysis' credibility and aligns it with established Cyber Security Standards.

5.7.1 Initial Scan Results — Vulnerability Detection

The initial Bandit scan produced 80,542 flagged items across the codebase; after filtering results from third party packages in the virtual environment, there were found to be 59 vulnerabilities in the project's own source files across 15 modules; the details of the 6 vulnerability categories are summarized in Table 2.

Example Evidence — Detected Findings

The following examples reproduce representative Bandit output entries as direct evidence of the detected issues. Each entry shows the tool-reported issue identifier, severity rating, CWE reference, and the corresponding source location. Figure 5, Figure 6, and Figure 7 represent the output from the Bandit simulation run of the Privex.

Table 2 Vulnerabilities Detected During Initial Bandit Scan

Vulnerability	CWE	Severity	File(s) Affected	Issues
Deprecated pyCrypto-library	CWE-327	High	api.py, dh_rsa.py, rsa.py, non_repudiation.py	28
Flask debug=True in production	CWE-94	High	app.py	1
Hardcoded credentials	CWE-259	Low	app.py	2
XSS via Markup() rendering	CWE-79	Medium	app.py	1
SQL injection via f-strings	CWE-89	Medium	todo_db.py	3
Weak RNG for crypto keys	CWE-330	Low	dh_rsa.py, test.py	2

Example 1: Flask Debug Mode (app.py, line 841)

```
>> Issue: [B105:hardcoded_password_string] Possible hardcoded
   password: 'pbrp vfiq jijb gqwf'
      Severity: Low          Confidence: Medium
      CWE:                                          CWE-259
   (https://cwe.mitre.org/data/definitions/259.html)  Location:
   .\app.py:526
 525     sender_email = "privex.official.contact@gmail.com"
 526     app_password = "pbrp vfiq jijb gqwf"
```

Fig. 5 Bandit output — Flask debug mode vulnerability (pre-remediation)

Example 2: Hardcoded Password String (app.py, line 526)

```
>> Issue: [B201:flask_debug_true] A Flask app appears to be
   run with debug=True, which exposes the Werkzeug
   debugger and allows the execution of arbitrary code.
      Severity: High          Confidence: Medium
      CWE:                                        CWE-94
   (https://cwe.mitre.org/data/definitions/94.html) Location:
   .\app.py:841
  840    init_db()
  841    socketio.run(app, debug=True)
```

Fig. 6 Bandit output — Hardcoded credential vulnerability (pre-remediation)

Example 3: SQL Injection Vector (todo_db.py, line 148)

```
>> Issue: [B608:hardcoded_sql_expressions] Possible SQL
   injection vector through string-based query construction.
      Severity: Medium        Confidence: Medium
      CWE:                                        CWE-89
   (https://cwe.mitre.org/data/definitions/89.html)
   Location: .\todo_db.py:148
   • c = conn.cursor()
   • c.execute(f'"UPDATE Tasks SET {set_clause} WHERE id = ?'", values)
```

Fig. 7 Bandit output — SQL injection vulnerability (pre-remediation)

5.7.2 Remediation — Code-Level Fixes Applied

Each detected vulnerability was resolved through targeted code modification. The following examples (through Table 3, Table 4, and Table 5) illustrate the before-and-after state of the corrected code for three representative cases.

Fix 1: Disabling Flask Debug Mode

Table 3 Code fix — Flask debug mode controlled via environment variable

Before Remediation	After Remediation

# app.py — line 841 (before) 	
socketio.run(app, debug=True)
	# app.py — line 841 (after)
	debug_mode = os.environ.get('FLASK_DEBUG','false') =='true' socketio.run(app,debug= debug_mode)

Fix 2: Removing Hardcoded Credentials

Table 4 Code fix — Secrets migrated to environment variables.

Before Remediation	**After Remediation**

# app.py (before) 	
app.secret_key = 'hagkfgkew$#%^%$hag'
app_password = 'pbrp vfiq jijb gqwf'
	# app.py (after)
	app.secret_key = os.environ.get('SECRET_KEY')
	app_password = os.environ.get ('GMAIL_APP_PASS')

Fix 3: Parameterised SQL Query

Table 5 Code fix — Dynamic SQL column names validated against allowlist

Before Remediation	**After Remediation**

# todo_db.py (before) 	
c.execute(f'"UPDATE Tasks SET {set_clause} WHERE id = ?'", values)
	# todo_db.py (after)
	ALLOWED = {'title','status','priority'} if not set(cols).issubset(ALLOWED):
	raise ValueError('Invalid column') c.execute(query, values)

5.7.3 Post-Remediation Validation Results

Following the application of all fixes, the Bandit scan was re-executed under identical configuration. The post-remediation scan confirmed that all six vulnerability classes identified in the initial analysis were fully resolved. Table 6 presents a comparison of pre- and post-remediation issue counts per category.

Table 6 Pre- and Post-Remediation Issue Count Comparison

Vulnerability	Pre-Remediation	Post-Remediation	Status
Deprecated pyCrypto library	28	0	Resolved
Flask debug=True in production	1	0	Resolved

Hardcoded credentials	2	0	Resolved
XSS via Markup() rendering	1	0	Resolved
SQL injection via f-strings	3	0	Resolved
Weak RNG for crypto keys	2	0	Resolved
TOTAL (source files)	37	0	Resolved

The validation process has been demonstrated in Table 6, which represents a complete security lifecycle, including automated detection, targeted remediation, and verified resolution, confirming that the system has zero security vulnerabilities as identified during initial security assessment and that it passes the security validation criteria for a production deployment. The validation results also support the integrity claims of the system as documented in this paper.

In Table 7, The security characteristics and cryptographic methods within Privex can be found in the Privex framework. Privex implements confidentiality at the highest level by encrypting data from end to end using AES encryption and subsequently encrypting the data using the Fernet encryption model. Because of this, only those who are intended recipients of the information will be able to decrypt the sent information, and thus, all other parties will not have access to the same information. In addition to confidentiality, Privex uses the HMAC and RSA signature algorithm to ensure the integrity of sent information; therefore, if any changes or tampering were made to send information at any point, the user would instantly know. To provide authentication, Privex uses a password hashing mechanism based on the SHA-256 hashing algorithm. Therefore, only registered users will have access to the cryptographic resources. Lastly, to provide non-repudiation, Privex offers a mechanism for creating a link between actions taken by users and the users themselves by using RSA digital signatures along with append-only audit logs. Therefore, the audit logs will link actions to an individual user, making it impossible for a user to deny an action taken.

6 Result

The length of time required to encrypt and decrypt each record, to rekey, and, also, compute the block response time is achieved through the use of Python's time module by recording a time stamp immediately before and after executing a test using time(). The delta (difference) between the ending time and beginning time is equal to the total execution duration in seconds. Additionally, by placing timing statements in the user simulation function, individual users' session times may be measured as well. Using this method

allows the accurate evaluation of the system's level of performance, response time, and scalability when subjected to concurrent user load.

Integrated Cryptographic & Session Memory Unit: The system requires an AMD Ryzen 5 5500U with Radeon Graphics paired with 16GB RAM to provide the computational multi-threading and memory capacity necessary to execute hybrid RSA-2048 and Diffie-Hellman operations while simultaneously managing the memory-hard scrypt algorithm for private key protection ; this specific hardware setup ensures the device can maintain stable encryption times (42.36 ms to 74.42 ms) and handle the significant rekey latency (up to 2864.89 ms) and blocking response times (up to 3778.4 ms) required for secure real-time sessions with up to 250 participants.

Table 8 presents the performance metrics of the proposed system under varying numbers of participants (10, 100, and 250). As the participant count increases, the rekey latency rises significantly from 209.20 ms to 2864.89 ms, indicating higher overhead during key updates in larger groups. Encryption time remains relatively stable, ranging between 42.36 ms and 61.58 ms, while decryption time shows a gradual increase from 2.05 ms to 8.3 ms. The blocking response time also increases with scale, from 3236.75 ms to 3778.4 ms, reflecting the additional processing and coordination required as the group size grows.

Table 7 Security Analysis table

Security Property	Mechanism Used in Privex	Effectiveness
Confidentiality	AES-based End-to-End Encryption with Multi-Fernet	Messages are encrypted at the sender and decrypted only by legitimate recipients. The server never accesses plaintext data, ensuring true end-to-end confidentiality.
Integrity	HMAC (via Fernet) + RSA Digital Signatures	Any modification of encrypted data is immediately detected. Digital signatures prevent message forgery or tampering.
Authentication	SHA-256 Password Hashing + Password-Protected RSA Private Keys	Ensures only authenticated users can access cryptographic keys and participate in communication.
Non-Repudiation	RSA Digital Signatures with Append-Only Audit Logs	Cryptographically binds messages to senders and prevents denial of message authorship.
Forward Secrecy	Diffie–Hellman Key Exchange with Rekeying	Past communications remain secure even if current keys are compromised.

Backward Secrecy	Key Regeneration on User Join/Leave/Block	New participants cannot decrypt previously exchanged messages.
Dynamic Access Control	Diffie–Hellman-Based Dynamic Blocking System	Immediately revokes access and invalidates keys of blocked users in real time.
Replay Attack Resistance	Session-Specific Keys with Automatic Rotation	Previously captured messages cannot be replayed after key updates.
Man-in-the-Middle Protection	RSA Authentication + Diffie–Hellman + TLS	Prevents interception, impersonation, and manipulation during key exchange and communication.
Key Leakage Protection	Encrypted Private Keys + Multi-Fernet-Based Key Distribution	Prevents exposure of complete session keys even if partial data is compromised.
02Insider Attack Mitigation	Digital Signatures and Per-User Key Shares	Legitimate users cannot impersonate others or access unauthorized session data.
Availability	Lightweight Cryptography and Co-Host Delegation	Ensures uninterrupted communication and eliminates single points of failure.

Table 8 Performance Metrics Across Varying Numbers of Participants

Number of Participants	Rekey Latency (ms)	Encryption Time (ms)	Decryption Time (ms)	Blocking Response Time (ms)
10	209.20	42.36	2.05	3236.75
100	1385.69	55.1	4.1	3556.68
250	2864.89	61.58	8.3	3778.4

The graphs in Figure 8 show that as the number of participants increases, rekey latency, decryption time, and blocking response time also increase due to higher system workload. In contrast, encryption time remains relatively stable with only minor variation. This indicates that the encryption process scales efficiently compared to other operations.

Table 9 shows the comparative performance analysis of Privex with the relates schemes with respect to the cryptographic primitive usage, architecture, etc. In comparing the features of the Privex proposed system with existing RSA-only and AES-only secure chat systems, previously published works like Singh et al. [15], as well as Singh et al. [16], are based solely on the AES symmetric encryption algorithm; while this algorithm performs well with encrypting messages, it does not provide adequate

strength in either key distribution or authentication methods to be considered a viable option. In comparison to those two studies, other systems like Carranza et al. [17] and Sönmez and Abbas [18] rely on a mix of both RSA and AES to provide both secure transmission of keys for authenticity and provide significant amounts of additional processing power for the secure transmission of both the keys for authenticity and the messages themselves.

Like the previous examples, Perkasa et al. [19], utilize Diffie Hellman along with AES, however they focus primarily on how they implement Diffie Hellman as a method of increasing the micro-service scalability as opposed to managing the dynamics associated with how to maintain the topography of a larger service. The Privex architecture, utilizing AES encryption, RSA digital signatures, and the Diffie-Hellman key exchange methodology, effectively creates a hybrid security model to optimally provide a high level of cryptographic security.

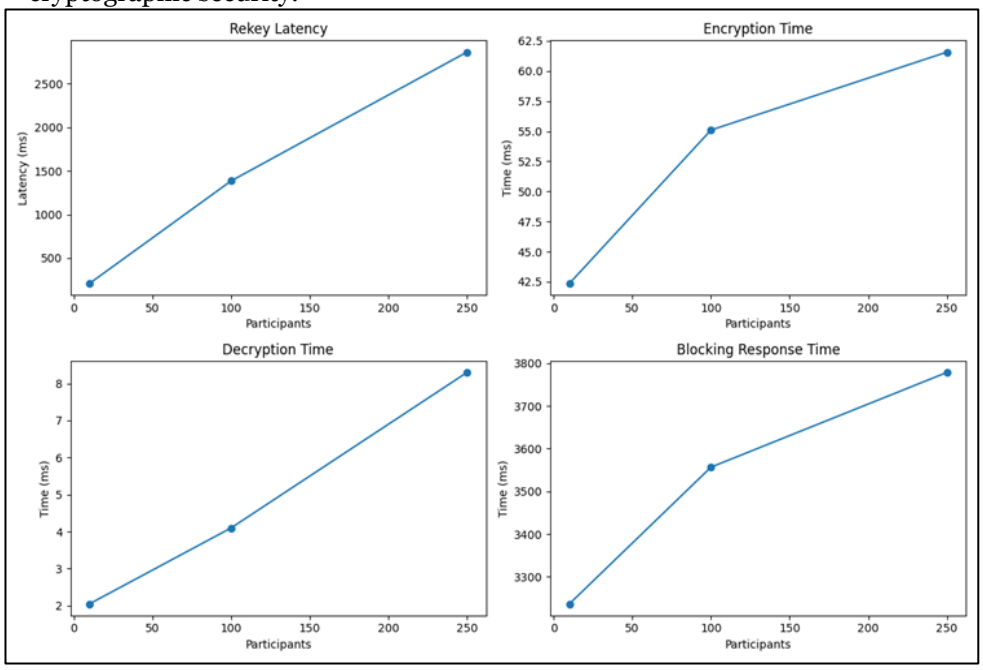

Fig. 8 Performance Analysis of Rekeying, Encryption, Decryption, and Blocking Operations with Increasing Participants.

AES quickly encrypts messages from one end to another while also providing the ability to authenticate, confirm that the private key is secure, and ensure non-repudiation using RSA. Diffie–Hellman provides an efficient way to create a new session key dynamically during an event (e.g. join/leave/blocking) so that a new session is always being created. This ensures both forward and backward secrecy. The proposed system is based on Python Flask (back-end), Flask-SocketIO (real-time client-server

communication), SQLite (session management), and allows for real-time efficient communication with secure key management.

In addition, Privex model is proven by the comparison charts to be more efficient operationally than the single algorithm method due to its application of only one algorithm to encrypt messages and signing (via RSA), and exchanging keys (via Diffie-Hellman) as well as less computation time than RSA only based systems while maintaining a greater level of assurance that the sender and receiver are who they claim to be compared to AES only based systems.

Finally, the dynamic re-keying and blocking functions supporting secure group communication do not significantly affect encryption/decryption performance. The performance charts and comparison table show that the Privex architecture will have significantly improved scalability, greater assurance of both security and performance than any currently available products.

Table 9 Comparison based on AES & RSA Efficiency, Key Exchange, and System Architecture.

Attribute	Singh et al. (2023) AES Android Chat [15]	Carranza et al. (2024) E2E Secure Chat [17]	Perkasa et al. (2025) DH+AES-256 Micro service [19]	Singh, Utama & Fat (2023) AES Chat Simulator [16]	Sönmez & Abbas (2017) RSA Client/ Server [18]	Privex
AES Usage	AES-128 for SMS & text message encryption	AES (CBC mode) for message encryption; session-specific keys.	AES-256 CBC for all message encryption	AES-256 in CBC mode for all chat messages	Not used; RSA-only system	AES via MultiFernet for E2E session encryption
RSA Usage	Not used (symmetric-only approach)	RSA for key exchange and key pair generation	RSA for digital signatures; key exchange via DH	Not used; symmetric-only system	RSA for both layers: client-server and client-client	RSA-2048 for digital signatures, non-repudiation, and private key protection
Key Exchange	Shared private key pre-established in app	RSA public key exchange then AES session key	Diffie-Hellman key exchange for shared secret	DH suggested for key exchange (future scope only)	RSA public key distribution between clients and server	Diffie-Hellman for dynamic session key exchange and blocking

Architecture	Android + SQLite; client-server	Python + tkinter GUI; client-server with sockets	Microservice: Nginx, WebSocket, Chat Service, RabbitMQ, PostgreSQL	Python CLI simulator; peer-to-peer via sockets	Client/server in MATLAB; two-layer RSA architecture	Python Flask + Flask SocketIO +SQLite; modular session-management backend
Order / Mode	Symmetric only; CBC mode implied	RSA (key exchange) then AES (message encryption); dual-layer	DH key exchange then AES-256 CBC encryption; scalable pipeline	Symmetric only; CBC mode for replay/MITM protection	Two-layer RSA: Layer 1 clients-server, Layer 2 clients-chat room	AES (E2E)+RSA (signatures) + DH (dynamic blocking); auto-rekeying on join/leave/block
Efficiency	Moderate - suitable for SMS-based mobile use; no asymmetric overhead	High - RSA adds overhead at setup but AES ensures fast real-time messaging	Very High - low latency, high throughput; benchmarked across 10-100,000 messages	Moderate - fast symmetric encryption; key sharing is main vulnerability	Low-Moderate - RSA-only is computationally expensive; small-scale use only	High - forward & backward secrecy; dynamic blocking; validated with Bandit static analysis

7 Conclusion

The proposed secure communication system combines various encryption systems to create an effective but simple communication tool. When employing AES, it employs the Multi-Fernet Encryption, to provide an end-to-end security system, keeping the data confidential and the keys in a safe location. Rather than using the symmetrical encryption only, it incorporates the RSA powered digital signatures to ensure that users can check the source of messages and avoid being denied of participation. Moreover, a Diffie-Hellman-based session control approach enhances user security since it allows administrators to block users, within no time, without disruption of ongoing chats. The Co-Host Delegation role enhances collaboration since it enables the transfer of control to be flawless throughout the meeting process, but it is extremely protective. It is tested and shows that the setup maintains high safety and at the same time does not compromise the convenience, fastness, or the desire to expand - which is perfect in virtual meetings, business correspondence, or collaborative work environments.

8 Future Scope

The proposed design may develop to meet the requirements of new trends in safety and convenience. Rather than the existing practices, the inclusion of

blockchain could be beneficial to store the session record and encryption evidence. Such a solution would result in an impeccable history that is more transparent. The other course of action would be to implement quantum resistant cryptography that would protect against the threats of powerful computers of the future. By adding AI-based anomaly detection, one can enhance the safety efforts by detecting bizarre activities - e.g., an unauthorized log-in or an internal risk - in real-time. To expand coverage, it might be relevant to address the system to different environments, such as mobile-native applications and minimal client installations, to expand usage by providing easier access. Speed maximization is important, especially when high-volume operations are involved, and smart key sharing and high-quality encryption reduce lag. Permission frameworks are better than broad rules because with permission frameworks it is much easier to control who can do what, usually by role or specific trait. Additionally, by combining encrypted cloud archives with zero-knowledge verification, it is possible to store the messages and documents safely over a long period of time, which suits the needs of larger organizations better.

Acknowledgment

We would like to take this opportunity to thank everyone who has helped to make this research project possible. We sincerely appreciate our supervisor and professor Dr Sudip Kumar Palit who has provided thorough and constant support through encouragement and provided great ideas and constructive feedback during our time together; his expertise has been essential to the success of this project as well as the decision on how the project would progress, the information required to complete the project and finally the conclusion of the project.

We would like to also thank our Institution UEM, Kolkata for providing the required materials, resources and support needed to complete this study. To Every author and peers, we appreciate the help in supporting through the collaboration, discussions and suggestions provided over our period of work together. We sincerely thank you all for the wonderful results of your effort and support through our time together.

Contribution of Each Author

Conception: Tiyasa Paul, Rohan Samanta, Pratik Saha, Partha Pratim Chaulia, Sudip Kumar Palit, Design of the work: Pratik Saha, Rohan Samanta, Investigation: Tiyasa Paul, Rohan Samanta, Data analysis and interpretation: Pratik Saha, Partha Pratim Chaulia, Drafting the article: Tiyasa Paul ,Partha Pratim Chaulia, Review editing of the article: Sudip Kumar Palit, Tiyasa Paul, Pratik Saha, Critical revision of the article: Sudip Kumar Palit, Final approval of the version to be published: Sudip Kumar Palit.

Conflict of Interest Statement

The authors declare no conflict of interests.

References

1. M. A. Jerlin, R. Shrivastav, K. Anusha, G. G. Devarajan, and S. M. Nagarajan, "Secure chat system: Harnessing the power of hybrid encryption for enhanced security and confidentiality," in *Proc. Int. Conf. Recent Trends Comput.*, Singapore: Springer, Jul. 2024, pp. 109–124.
2. M. A. Almaiah, Z. Dawahdeh, O. Almomani, A. Alsaaidah, A. Al-Khasawneh, and S. Khawatreh, "A new hybrid text encryption approach over mobile ad hoc network," *Int. J. Electr. Comput. Eng.*, vol. 10, no. 6, pp. 6461–6471, 2020.
3. Y. Fouzar, A. Lakhssassi, and M. Ramakrishna, "A novel hybrid multikey cryptography technique for video communication," *IEEE Access*, vol. 11, pp. 15693–15700, 2023.
4. L. Harn and C.-F. Hsu, "A practical hybrid group key establishment for secure group communications," *Comput. J.*, vol. 60, no. 11, pp. 1582–1589, 2017.
5. S. Ahmad, S. Mehfuz, and J. Beg, "Hybrid cryptographic approach to enhance the mode of key management system in cloud environment," *J. Supercomput.*, vol. 79, no. 7, 2023.
6. J. Sivakumar and S. Ganapathy, "An effective data security mechanism for secured data communications using hybrid cryptographic technique and quantum key distribution," *Wireless Pers. Commun.*, vol. 133, no. 3, pp. 1373–1396, 2023.
7. M. A. Jerlin, R. Shrivastav, K. Anusha, G. G. Devarajan, and S. M. Nagarajan, "Secure chat system: Harnessing the power of hybrid encryption for enhanced security and confidentiality," in *Proc. Int. Conf. Recent Trends Comput.*, Singapore: Springer, Jul. 2024, pp. 109–124.
8. K. Bhatele, A. Sinhal, and M. Pathak, "A performance-centric comparative study of hybrid security protocol architectures," in *Proc. AISOBE 2012*, India: Springer, 2012.
9. A. M. Abdullah, "Advanced encryption standard (AES) algorithm to encrypt and decrypt data," *Cryptography and Network Security*, vol. 16, no. 1, p. 11, 2017.
10. D. K. Zala and M. A. Khan, "Compressive method for securing text data using Base64 encoding and Fernet cryptography," in *Proc. 2nd Int. Conf. Sustainable Comput. Smart Syst. (ICSCSS)*, IEEE, 2024.
11. R. Oppliger, *SSL and TLS: Theory and Practice*. Norwood, MA, USA: Artech House, 2023.
12. J. H. Seo, "Efficient digital signatures from RSA without random oracles," *Inf. Sci.*, vol. 512, pp. 471–480, 2020.
13. F. Affan and R. Rehan, "Secure chat application end-to-end encryption," *The Multidisciplinary Edge: Advancing Academic Research & Development*, p. 138, 2025.

14. D. L. Hoang and T. L. Tran, “New proofs for pseudorandomness of HMAC-based key derivation functions (RFC 5869),” *J. Inf. Secur. Appl.*, vol. 93, p. 104179, 2025.
15. S. Singh, H. S. Utama, and J. Fat, “Chat simulator using AES encryption,” *Int. J. Appl. Sci. Technol. Eng.*, vol. 1, no. 3, pp. 931–940, 2023.
16. V. Singh, S. Janarthanan, U. K. Roshan, and N. Vaid, “A secure web-based Android chat application using the AES encryption algorithm,” in Proc. 2023 5th International Conference on Advances in Computing, Communication Control and Networking (ICAC3N)., Dec. 2023.
17. H. Carranza, M. Bustamante, A. Carranza, and S. Muniganti, “Secure chat application with an end-to-end encryption.” In *Proceedings of the World Congress on Electrical Engineering and Computer Systems and Science*. Avestia Publishing. doi:10.11159/cist24.140.
18. F. Sönmez and M. K. Abbas, “Development of a client/server cryptography-based secure messaging system using RSA algorithm,” *J. Manag. Eng. Inf. Technol.*, vol. 4, no. 6, 2017.
19. M. J. P. Perkasa, H. M. Ramdan, A. J. Maliki, and Somantri, “Implementation of secure end-to-end encrypted chat application using Diffie–Hellman key exchange and AES-256 in a microservice architecture,” *Eng. Proc.*, vol. 107, no. 1, p. 98, 2025.

Tiyasa Paul is a final-year B. Tech student in Computer Science and Engineering, specializing in Artificial Intelligence and Machine Learning at University of Engineering and Management Kolkata, India. Her undergraduate studies began with completing Secondary and Higher Secondary Education at Maria's Day School, Howrah. She has done work related to artificial intelligence and possess a strong interest of research in this field of study.

Rohan Samanta is a final-year B. Tech student in Computer Science and Engineering, specializing in Artificial Intelligence and Machine Learning at University of Engineering and Management Kolkata, India. He attended Lakshya High School throughout his education history and has a profound interest in emerging technology and software development careers.

Pratik Saha is a final-year B. Tech student in Computer Science and Engineering, specializing in Artificial Intelligence and Machine Learning at University of Engineering and Management Kolkata, India. He went to Bidhannagar Government High School for all of his schooling history, and is seeking to create practical applications that utilise both artificial intelligence and machine learning technologies.

Partha Pratim Chaulia is a final-year B. Tech student in Computer Science and Engineering, specializing in Artificial Intelligence and Machine Learning at University of engineering and Management Kolkata, India. He completed his school education in Secondary Education at Ramakrishna Mission Vidyabhavan and completed his study for Higher Secondary Education at Jafala Adarsh Vidyayatan; and has an interest in Intelligent Systems and creating innovative solutions.

Sudip Kumar Palit is an Associate Professor and Centre-in-Charge of the Cybersecurity Centre of Excellence at the University of Engineering and Management (UEM), Kolkata. He received his Ph.D. in Computer Science and Engineering from UEM in 2025. With 15 years of teaching and 10 years of research experience, his research interests include cybersecurity, cryptography, blockchain, SOC-SIEM frameworks, performance analysis of blockchain systems, and authentication protocols. He actively mentors' students in applied cybersecurity research and development.