IJMSERH

International Journal of Multidisciplinary and Scientific Emerging ResearcH (IJMSERH)

Volume 13, Issue 2, April-June 2025

Impact Factor: 9.274

ijmserh@gmail.com

www.ijmserh.com

International Journal of Multidisciplinary and Scientific Emerging ResearcH (IJMSERH)

| ISSN: 2349-6037 | www.ijmserh.com ||Impact Factor: 9.274||Quarterly, Peer Reviewed & Refereed Journal|

|| Volume 13, Issue 2, April-June 2025 ||

DOI: 10.15662/IJMSERH.2025.1302037

Key Agreement for Group Data Sharing in Cloud Computing Using Block Design

K. Kishore

Ranipetai Engineering College, Tamil Nadu, India

ABSTRACT: The utilization of cloud computing for data sharing facilitates collaborative work and offers extensive practical applications. However, ensuring data security and efficient sharing within groups presents significant challenges. Key agreement protocols play a crucial role in addressing these challenges. This paper introduces a novel key agreement protocol based on symmetric balanced incomplete block design (SBIBD) to support multiple participants in cloud environments. The protocol's structure allows for flexible scalability according to the block design. We present formulas for generating a common conference key K for multiple participants, leveraging the properties of the block design to manage computational complexity and communication efficiency effectively. Moreover, our protocol exhibits fault tolerance, enhancing data sharing resilience against key attacks.

KEYWORDS: Key Agreement Protocol, Symmetric Balanced Incomplete Block Design (SBIBD), Security, Data Sharing, Cloud Computing.

I. INTRODUCTION

In cloud computing environment, ensuring data sharing among multiple users within a group is quite challenging as it is not secure. Current approaches often lack efficiency and scalability while maintaining robust security measures. We want to build a mechanism which facilitates secure communication and enables seamless access to shared data within the cloud environment for a group.

Cloud computing and cloud storage have emerged as crucial developments, transformed various aspects of modern life and significantly enhanced productivity. The preference for storing diverse data types on cloud servers is driven by the need for convenient access and the avoidance of local infrastructure overheads for companies and organizations. However, this reliance on cloud storage introduces security challenges, including potential attacks from malicious users and cloud providers. To address these concerns, several schemes, such as those proposed by Zhou, Varadharajan, Hitchens [2] and Chen et al. [3], have focused on preserving the privacy of outsourced data. While these schemes have addressed security concerns for individual data owners, there is a growing demand for protocols that support secure group data sharing under cloud computing environments.

A key agreement protocol plays a crucial role in ensuring secure and efficient data sharing among multiple participants in cloud computing. Originating from the seminal work of Diffie-Hellman, key agreement protocols enable the generation of a common conference key to facilitate secure communication. However, traditional protocols like Diffie-Hellman lack authentication mechanisms and are limited to supporting communication between two participants. Efforts to enhance the security and scalability of key agreement protocols, as evidenced by Law et al. [4] and Yi [5], have focused on integrating authentication mechanisms and addressing the limitations of traditional protocols to support multiple participants. Additionally, ongoing research endeavours, as covered in the literature [6]-[9], aim to improve the security and communication efficiency of key agreement protocols, reflecting the continuous evolution of cryptographic techniques in cloud computing security. Inspired by research on load balancing algorithms utilizing block design, such as Chung and Bae [10] and Lee et al. [11], this work introduces the symmetric balanced incomplete block design (SBIBD) in designing key agreement protocols. The integration of SBIBD aims to reduce the complexity of communication and computation while ensuring robust security measures. This novel approach represents a significant contribution to the field, as it addresses the need for efficient and secure group data sharing protocols within cloud computing environments, laying the foundation for further advancements in cloud security.

1.1 Main Contributions

The paper introduces a novel block design-based key agreement protocol aimed at enhancing the security and efficiency of data sharing among multiple participants. By extending the structure of Symmetric Balanced Incomplete Block Designs (SBIBD), the protocol facilitates secure communication among data owners, enabling them to freely

| ISSN: 2349-6037 | www.ijmserh.com ||Impact Factor: 9.274||Quarterly, Peer Reviewed & Refereed Journal|

|| Volume 13, Issue 2, April-June 2025 ||

DOI: 10.15662/IJMSERH.2025.1302037

share outsourced data. Notably, SBIBD serves as the foundational model for group data sharing, specifically tailored to support collaborative data sharing in cloud computing environments.

The protocol's utilization of SBIBD expands its application scope, enhancing security and flexibility. Additionally, it achieves reduced communication complexity without added computational overhead, with communication complexity of $O(n\sqrt{n})$ and computational complexity of $O(nm^2)$, where n represents the participant count and m denotes the extension degree of the finite field Fpm. Let's discuss some main contributions that we want to share in this paper:

- *Data sharing model for Symmetric balanced incomplete block design:* The protocol leverages SBIBD to establish a robust framework for secure key agreement, enabling efficient communication among multiple participants. By incorporating SBIBD's structure, the protocol enhances security while ensuring data owners can share outsourced data with confidence.

- *Features de-centralized key agreement*: The protocol offers authentication services to verify the identity of participating entities, bolstering the overall security of the data sharing process. Moreover, it integrates fault tolerance mechanisms to mitigate the impact of participant failures or malicious activities, by ensuring uninterrupted communication and data sharing.

- *Intended in a Cloud Computing scenario* The protocol facilitates secure group data sharing in the cloud, where participants jointly generate a shared key for accessing outsourced data. Utilizing SBIBD methods, it safeguards against unauthorized access, bolstering overall data sharing security.

1.2 Structural Overview

We want to organize the rest of the paper as follows: In section 2 we can be able to see system architecture and process. Section 3 has a brief explanation about group data sharing. In section 4 we will get to know about Key Agreement Protocol. Section 5 contains the Security Analysis, and Section 6 deals with our Implementation and Performance Evaluation. Sections 7 and 8 completes the conclusion part and references.

II. SYSTEM OVERVIEW

In combinatorial mathematics, block designs serve as fundamental structures comprising a set and subsets, chosen to fulfill specific criteria relevant to various applications. The balanced incomplete block design (BIBD), detailed in Definition 3, delineates conditions under which an incidence structure s = (V, B) qualifies as a BIBD or a symmetric BIBD (SBIBD). Central to this framework are parameters such as v (the number of elements), b (the number of blocks), k (the elements per block), r (the occurrences of elements in blocks), and λ (the pairwise block intersections). By requiring a (v, k + 1, 1)-design, where k is prime and λ = 1, our decentralized group data sharing model is constructed, facilitating effective information exchange among participants in a key agreement protocol. Leveraging the properties of this design, each participant can efficiently determine message recipients or senders, forming the cornerstone of our protocol's information exchange mechanism.

Here is the definition that mentioned in the paper about the structure of the block design based on SBIBD.

> ***Definition 1.*** *Let V = (0, 1, 2... v-1) be a set of v elements and B = {B_0, B_1, B_2 ... B_{b-1}} be a set of b blocks, where Bi is a subset of V and $|B_i| = k$. For a finite incidence structure S = (V, B), if s satisfies the following conditions, then it is a BIBD, which is called a (b, v, r, k, λ)-design.*
>
> 1. *Each element of V appears in exactly r of the b blocks.*
> 2. *Every two elements of V appear simultaneously in exactly λ of the b blocks.*
> 3. *Parameters k and v of V meet the condition of k<v. Thus, no block contains all the elements of set V.*
> 4. *Parameters b and v of V meet the condition of $b \geq v$. The case of equality is called a symmetric design.*

Essentially, "A TPA, cloud and users are involved in the model, where the TPA is responsible for cloud storage auditing, fault detection and generating the system parameters. Cloud, who is a semi-trusted party, provides users with data storage services and download services. Users can be individuals or staff in a company. To work together, they form a group, upload data to the cloud server and share the outsourced data with the group members. In practice, users can be mobile Android devices, mobile phones, laptops, nodes in underwater sensor networks and so forth."1

International Journal of Multidisciplinary and Scientific Emerging ResearcH (IJMSERH)

| ISSN: 2349-6037 | www.ijmserh.com ||Impact Factor: 9.274||Quarterly, Peer Reviewed & Refereed Journal|

|| Volume 13, Issue 2, April-June 2025 ||

DOI: 10.15662/IJMSERH.2025.1302037

For this project, users are a client application on some desktop device, which can send messages which are routed to through the cloud - where the information is exchanged and can be accessed by other users/clients. The symmetric balanced incomplete block design (SBIBD) key agreement protocol is used during the info exchange. The info exchange is valid only for the users in a specified group, and external entities including the cloud provider would not be able to participate or read the exchanged information.

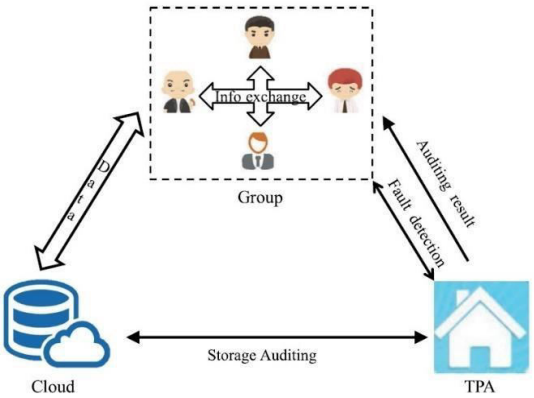

Fig 1: Original System model of data sharing in cloud computing.

2.1 System Architecture

It is stated, in the context of SBIBD, that a TPA isn't required: "the group data sharing model is based on the SBIBD, where a trusted third party is not required. With respect to this model, all the participants exchange messages from intended entities according to the structure of the SBIBD to determine a common conference key."1 That is, we use a decentralized model where the participants themselves generate a common conference key through the proposed SBIBD protocol.

Due to this, our implementation does not bring a TPA into consideration.

Fig 2 represents our refined system architecture that we aim to follow for the implementation of this project.

In this architecture, an Amazon EC2 instance serves as the backbone, hosting a server application that interacts with clients. The server application communicates with an SQLite database, managing data efficiently for the user group.

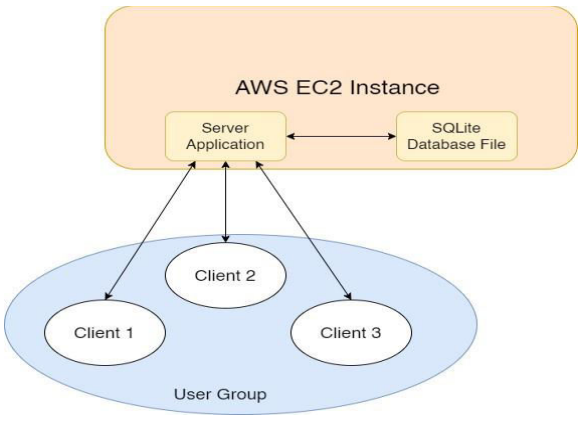

Fig 2: Refined System model of data sharing in cloud computing.

Three clients that are representing users and send requests to the server application, which processes them accordingly. This setup enables seamless interaction between clients and the server, facilitating tasks such as data retrieval and updates. The lightweight nature of SQLite ensures optimal handling of read and write operations. Overall, this architecture showcases the scalability and efficiency of cloud computing in managing applications and serving user needs.

| ISSN: 2349-6037 | www.ijmserh.com ||Impact Factor: 9.274||Quarterly, Peer Reviewed & Refereed Journal|

|| Volume 13, Issue 2, April-June 2025 ||

DOI: 10.15662/IJMSERH.2025.1302037

We've also created another diagram to demonstrate the system's resilience against potential bad actors, specifically considering the scenarios laid out in the paper.

Here in this system model, which is shown in Fig 3, we can see a robust security framework within an AWS EC2 instance hosting a server application. It addresses both insider and outsider threats by implementing encryption and authentication measures. Encryption ensures data confidentiality during transmission, thwarting unauthorized access by cloud providers or other insiders. For external threats, a symmetric key agreement protocol and authentication process restrict access to authorized clients only. Additionally, the system adopts a multilayered security approach, incorporating firewalls, encrypted tunnels, and access control mechanisms to fortify its defences. By combining these strategies, the system builds resilience against potential attacks, safeguarding data integrity and maintaining operational continuity.

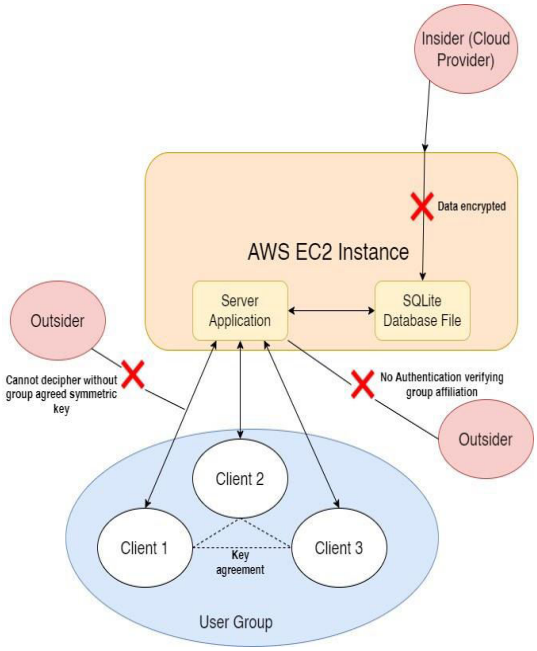

Fig 3: System model resilience against potential bad actors.

2.2 System Process

We want to build a system process flowchart where we want to send encrypted messages, ensuring secure communication between users and the server. It begins with user registration, verifying if the user is already registered in the system. If not, the user must complete the registration process to gain access. Subsequently, upon successful login, the system checks if a key agreement has been established between the user and the server. If not, the key agreement process is initiated, facilitating the creation of a symmetric key shared between the user and the server. This key becomes pivotal for encryption and decryption during message transmission.

Once the key agreement is confirmed, users can proceed to send messages securely. They encrypt their messages using the agreed-upon symmetric key before transmission. The encrypted message travels to the server, where it is received and decrypted using the same symmetric key. The server then processes the decrypted message and stores it in the database for persistence. Throughout this process, clients can make requests to the server for various interactions, ensuring seamless communication. By incorporating these steps, the system maintains a robust security framework, safeguarding sensitive information and ensuring that only authorized users can access and send encrypted messages within the system.

International Journal of Multidisciplinary and Scientific Emerging ResearcH (IJMSERH)

| ISSN: 2349-6037 | www.ijmserh.com ||Impact Factor: 9.274||Quarterly, Peer Reviewed & Refereed Journal|

|| Volume 13, Issue 2, April-June 2025 ||

DOI: 10.15662/IJMSERH.2025.1302037

Here is the system process for our project.

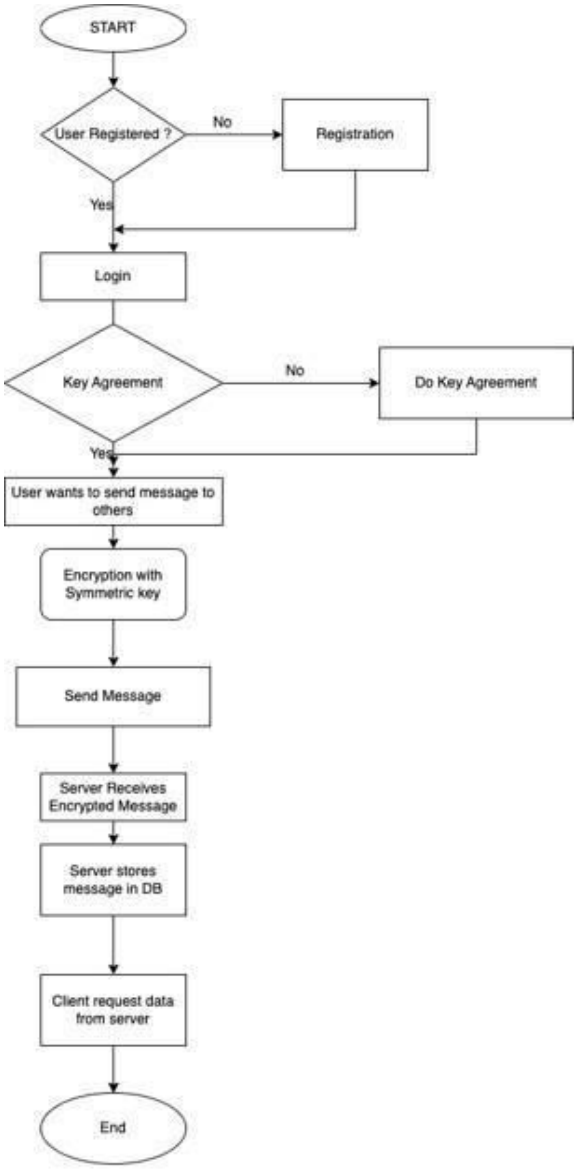

Fig 4: Flowchart of System Process

III. BLOCK DESIGN

The Block Design-based group data sharing model leverages the structure of a (v, k + 1, 1)-design, derived from a symmetric balanced incomplete block design (SBIBD). This model facilitates efficient key agreement protocols among (v) participants by ensuring each participant is represented (k + 1) times within blocks, enabling comprehensive pairwise interactions. So, to construct this block design we want to implement an algorithm. Through transformations tailored to the specific requirements of the group data sharing scenario, the initial design is refined to establish a robust framework where participants can securely collaborate and share information. By harnessing the inherent properties of block designs, such as balanced representation and pairwise coverage, this model offers a structured approach to facilitate secure and scalable data sharing among multiple participants.

3.1 Construct (V, K+1, 1) – Design

In our group data sharing model, the SBIBD parameters hold significant implications. A (v, k + 1, 1)-design entails v denoting both participants and blocks, each block accommodating k + 1 participants, with each participant appearing k

| ISSN: 2349-6037 | www.ijmserh.com ||Impact Factor: 9.274||Quarterly, Peer Reviewed & Refereed Journal|

|| Volume 13, Issue 2, April-June 2025 ||

DOI: 10.15662/IJMSERH.2025.1302037

+ 1 times across all v blocks. Additionally, every pair of participants simultaneously appears in precisely one of the v blocks. Algorithm 1, as outlined in previous works, is instrumental in constructing the (v, k + 1, 1)-design. The algorithm initially selects a prime number k, from which the number of participants, v, is computed as ($v = k^2 + k + 1$). Subsequently, utilizing modular operations and systematic computations, Algorithm 1 outputs the structure of the (v, k + 1, 1)-design, with each block represented by numbers $B_{i,j}$ for ($i = 0, 1, ..., k^2 + k$) and ($j = 0, 1, ..., k$).

The SBIBD parameters and Algorithm 1 facilitate the creation of a (v, k + 1, 1)-design tailored to accommodate v participants efficiently. This algorithmic approach not only determines participant involvement in each block but also ensures the satisfaction of key conditions of the (v, k + 1, 1) design, wherein each participant appears k + 1 times and each pair of participants appears exactly once. Leveraging these properties, our group data sharing model optimizes communication costs within the proposed protocol, enhancing its scalability and efficiency. Detailed insights into protocol implementation and performance evaluation based on this model are provided in Sections 4 and 6, respectively.

In Algorithm 1, the symbol MOD_k denotes the modular operation, which computes the class residue as an integer within the range of 0 to k - 1. This algorithm serves as the foundation for constructing a (v, k + 1, 1)-design involving v participants. Notably, Algorithm 1 facilitates the direct determination of participant allocation within each block. For instance, in considering a (13, 4, 1)-design involving 13 participants, the participant to be included in the 3rd column of the 8th block can be determined through computation This calculation involves applying the modular operation to specific parameters within the algorithm, ensuring precise participant assignment with the designed structure.

$$B_{7,2} = jk + 1 + MOD_k\,(i - j + (j - 1)\lfloor (i-1)/k \rfloor)$$
$$= 2*3 + 1 + MOD_3(7 - 2 + (2 - 1)\,\lfloor (7-1)/3 \rfloor)$$
$$= 7 + MOD_3(5 + 1*2)$$
$$= 7 + 1 = 8$$

Therefore, based on this calculation, it is determined that participant 8 is to be included in the 3rd column of the 8th block. Here, i denotes the i^{th} participant.

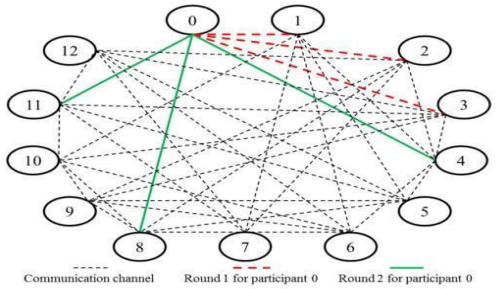

Fig 6. (13, 4, 1)-design group data sharing model

Algorithm 1: Generation of a (v, k+1, 1)-design

for $i = 0;\ i \leq k;\ i{+}{+}$ *do*
  for $j = 0;\ j \leq k;\ j{+}{+}$ *do*
    if $j == 0$ *then*
      $B_{i,j} = 0;$
    else
      $B_{i,j} = ik + j;$
    end if
  end for
end for
for $i = k + 1;\ i \leq k^2 + k;\ i{+}{+}$ *do*
  for $j = 0;\ j \leq k;\ j{+}{+}$ *do*
    if $j == 0$ *then*
      $B_{i,j} = \lfloor (i-1)/k \rfloor;$
    else
      $B_{i,j} = jk + 1 + MOD_k(i - j + (j - 1)\lfloor (i-1)/k \rfloor);$
    end if
  end for
end for

| ISSN: 2349-6037 | www.ijmserh.com ||Impact Factor: 9.274||Quarterly, Peer Reviewed & Refereed Journal|

|| Volume 13, Issue 2, April-June 2025 ||

DOI: 10.15662/IJMSERH.2025.1302037

IV. KEY AGREEMENT PROTOCOL

Key agreement protocols are cryptographic techniques designed to securely establish a shared secret key between multiple parties for encrypted communication. In the context of group data sharing, these protocols are essential for ensuring confidentiality and integrity of exchanged information among multiple users. Asymmetric key agreement protocols, such as Diffie-Hellman or RSA, utilize a pair of keys which are public and private for secure communication. They offer a mechanism for parties to establish a shared secret over an insecure channel without prior communication. Conversely, symmetric key agreement protocols, like AES or DES, employ a single shared secret key for encryption and decryption. These protocols are essential in group data sharing scenarios as they ensure secure communication among multiple parties.

4.1 Key Generation

To generate a key and distribute it among multiple clients, we employ a symmetric key agreement protocol and want to calculate the time it takes to distribute the key and the time it takes to distribute the symmetric key to all clients is dependent on several factors. Initially, when the first client (the admin) connects to the server, they generate an AES symmetric key for message encryption. As subsequent clients join the group, the admin (Client 1) initiates a Diffie-Hellman exchange with each client individually, enabling secure communication between them. Once this exchange is complete, the admin can securely send the AES key to each client, enabling them to encrypt and decrypt messages. This process of distributing the key to each client, including the DiffieHellman handshakes and key exchanges, is included in the overall time duration.

The time it takes for this symmetric key distribution process to complete depends on the number of clients in the group and the efficiency of the network and cryptographic operations. Each additional client joining the group necessitates another round of DiffieHellman exchange and key distribution, potentially increasing the overall time required. However, once all intended clients have received the AES key, the timer stops, indicating that the symmetric key has been successfully distributed to all participants, and secure communication within the group can commence.

```java
public class KeyGeneration {

    // Method to generate a symmetric key
    static SecretKey generateSymmetricKey() throws NoSuchAlgorithmException {

        // Create a KeyGenerator instance for AES (Advanced Encryption Standard)
        KeyGenerator keyGen = KeyGenerator.getInstance("AES");

        // Initialize the KeyGenerator
        keyGen.init(128); // Use 128 bits key size for AES
        // Generate the symmetric key

        return keyGen.generateKey();
    }

    public static void main(String[] args) {
        try {
            // Generate symmetric key
            SecretKey key = generateSymmetricKey();
            System.out.println("AES Key: " + key);

        } catch (NoSuchAlgorithmException e) {
            e.printStackTrace();
        }
    }
}
```

Fig 7: Generating a Key

V. SECURITY ANALYSIS

In this section, we prove that our protocol is secure against passive attacks and active attacks.

5.1 Against Passive

In the proposed key agreement protocol for group data sharing, security against passive attacks is ensured through various cryptographic measures. Participants and volunteers in the protocol, along with the adversary, are modeled as probabilistic polynomial time Turing machines. While the adversary has access to system parameters and public keys, the secret key and ephemeral key of each participant remain secure due to the hardness of the integer factorization problem and the elliptic curve discrete logarithm problem (ECDLP), respectively. The protocol's security hinges on the polynomial indistinguishability between two tuples of random variables, ensuring that the probability of distinguishing them is negligible for all polynomial-time distinguishers. With the security parameter 'l' defining the key agreement

| ISSN: 2349-6037 | www.ijmserh.com ||Impact Factor: 9.274||Quarterly, Peer Reviewed & Refereed Journal|

|| Volume 13, Issue 2, April-June 2025 ||

DOI: 10.15662/IJMSERH.2025.1302037

protocol's security, all algorithms operate within probabilistic polynomial time, thereby providing robust protection against passive attacks.

5.2 Active

The key agreement protocol is fortified against active attacks through various mechanisms. It thwarts key compromise impersonation by ensuring independence among participants' long-term secret keys and tying signatures with timestamps, preventing replay attacks. Furthermore, the protocol's session keys are randomly generated for each session, ensuring resilience against known session key attacks. Perfect forward security is achieved, as compromising long-term keys does not compromise previous session keys. The protocol also resists different key attacks through fault detection mechanisms and provides key confirmation, ensuring participants possess the common conference key. Additionally, it mitigates denial of service attacks by enforcing participant removal for failure to resend fault reports. By leveraging these measures, the protocol maintains robust security, ensuring secure communication among users and the server in the proposed system architecture.

VI. IMPLEMENTATION AND PERFORMANCE EVALUATION

Our implantation scenario is a group messaging application – where multiple remote users can use a cloud computing environment to send messages to one another for the purpose of communication. We use the principles covered previously in this paper to implement a sample scenario where symmetric key agreements are a useful tool for ensuring security and privacy for users' messages.

Section 4.1 describes the key agreement process for distributing a symmetric AES key securely to the group participants. Figure 7 below demonstrates our implementation of creating a secure channel for transporting the encryption (AES) to other clients, based on the principles of the Diffie-Hellman algorithm.

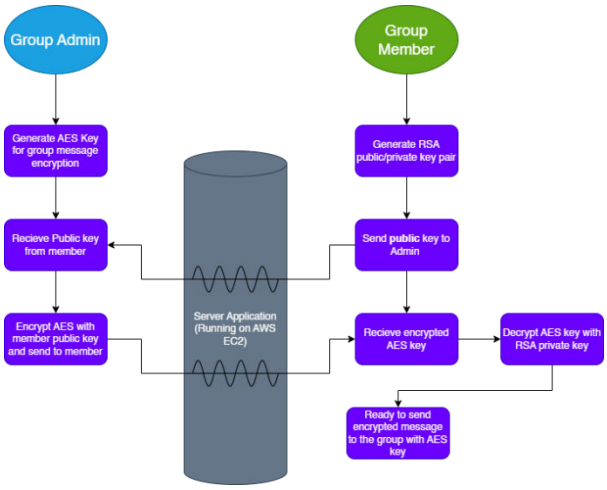

Fig 8. AES Key encryption with RSA

The group admin generates a universal symmetric key (using the AES algorithm) that all members of the group must use to encrypt and decrypt messages. But how will the administrator send the key over an insecure channel, through the EC2 server, for the very purpose of securing future messages? A man in the middle, can access the key during transport – letting them read any message, even if encrypted. Hence, whenever an authorized client connects to the server, it sends a request for the secure key, i.e. the AES key, from the admin – along with this request, it sends its own generated public/private key pair, using the RSA algorithm. The public key can be accessed and used by anyone to encrypt a message – but only the private key can decrypt that message, not the public key.

Once the admin received the aspiring group member's RSA public key, they encrypt the AES key itself with the public key and send it over the insecure channel back to the source client – the client is then able to decrypt the AES key using its own private RSA key. This allows the client/new group member to now encrypt and decrypt the message it sends and receives respectively.

| ISSN: 2349-6037 | www.ijmserh.com ||Impact Factor: 9.274||Quarterly, Peer Reviewed & Refereed Journal|

|| Volume 13, Issue 2, April-June 2025 ||

DOI: 10.15662/IJMSERH.2025.1302037

6.1 Implementation

We shall now observe the implementation of this process – Figures 9-13 demonstrate the methods that make this process possible. Java 21 was used for the implementation of this code.

```
// Generating public and private keys using RSA algorithm.
public void generateRSAKeyPair() throws Exception {

    // Generate the private/public key only once during each run
    if (this.privateKey == null && this.publicKey == null) {
        SecureRandom secureRandom = new SecureRandom();

        KeyPairGenerator keyPairGenerator = KeyPairGenerator.getInstance(RSA);

        keyPairGenerator.initialize(2048, secureRandom);

        keypair = keyPairGenerator.generateKeyPair();

        // System.out.println("Public Key is: " +
        // DatatypeConverter.printHexBinary(keypair.getPublic().getEncoded()));
        // Convert the encrypted message to Base64 encoded string
        publicKey = Base64.getEncoder().encodeToString(keypair.getPublic().getEncoded());
        log.debug("Public Key is:" + publicKey);
        // System.out.println("Public Key is:" + publicKey);

        // System.out.println("Private Key is: " +
        // DatatypeConverter.printHexBinary(keypair.getPrivate().getEncoded()));
        privateKey = Base64.getEncoder().encodeToString(keypair.getPrivate().getEncoded());
        log.debug("Private Key is:" + privateKey);
        // System.out.println("Private Key is:" + privateKey);
```

Fig 9: Generating RSA Key pair on client

In Figure 9, we use the KeyPairGenerator class from the java.Security package. We initialize it to utilize the RSA algorithm and generate the public and private key pair and save it to the KeyPair object. On the client's log, the keys are displayed as strings.

```
// Encryption function which converts
// the plainText into a cipherText
// using private Key.
public String do_RSAEncryptionKey(String plainText, PublicKey publicKey) throws Exception {
    Cipher cipher = Cipher.getInstance("RSA/ECB/PKCS1Padding");
    cipher.init(Cipher.PUBLIC_KEY, publicKey);
    byte[] cipherText = cipher.doFinal(plainText.getBytes());
    return (Base64.getEncoder().encodeToString(cipherText));
}
```

Fig 10: Performing RSA Encryption on ARS Key

This method takes a public key of type RSA, and encrypts some text to ciphertext, and returns the result as a string to the calling method.

```
} else if (message.getMessageType() == MessageType.PROCESS_SECRET_KEY) {
    /*
     * String clientPublicKey = (String) message.getMessageContent();
     * adminReply.setClientId(server.getAdminClientId());
     * adminReply.setReplyToclientId(clientId);
     * adminReply.setMessageContent(clientPublicKey); adminReply.setMessageDate(new
     * Date()); adminReply.setMessageType(MessageType.PROCESS_SECRET_KEY);
     */
    Integer replyClientId = message.getReplyToclientId();
    log.debug("Reply to Client Id " + replyClientId);
    String clientPublicKeyStr = (String) message.getMessageContent();
    PublicKey clientPublicKey = new KeyPairFromString().parsePublicKey(clientPublicKeyStr);
    String encryptedSecretKey = encryption.do_RSAEncryptionKey(this.mySecretKey, clientPublicKey);

    Message secretMessage = new Message();
    secretMessage.setReplyToclientId(replyClientId);
    secretMessage.setMessageDate(new Date());
    secretMessage.setMessageType(MessageType.SECRET_KEY);
    secretMessage.setMessageContent(encryptedSecretKey);
    this.sendMessage(secretMessage);
```

Fig 11: Encrypting AES Client Key (on admin side) using a particular client's public key

We can see in this method, that it calls the function demonstrated in Figure 10. The client's public key that the admin received is passed in as a PublicKey object and the AES key is passed in as a plaintext string – to the RSA encryption method. After the encryption process is completed, it is sent as a secret message to the client.

```
public String do_RSADecryptionKey(String cipherText, PrivateKey privateKey) throws Exception {
    byte[] encryptedSecretKeyDecoded = Base64.getDecoder().decode(cipherText);
    Cipher cipher = Cipher.getInstance("RSA/ECB/PKCS1Padding");
    cipher.init(Cipher.PRIVATE_KEY, privateKey);
    byte[] result = cipher.doFinal(encryptedSecretKeyDecoded);
    return new String(result);
}
```

Fig 12: Performing RSA Decryption on encrypted AES Key

In Figure 12, the ciphertext is received as a string, and the RSA type private key. The ciphertext is then decrypted using the private key and plaintext returned as a string to the calling method.

| ISSN: 2349-6037 | www.ijmserh.com ||Impact Factor: 9.274||Quarterly, Peer Reviewed & Refereed Journal|

|| Volume 13, Issue 2, April-June 2025 ||

DOI: 10.15662/IJMSERH.2025.1302037

```
} else if (message.getMessageType() == MessageType.ACCEPT_SECRET_KEY) {
    /*
     * Message reply = new Message();
     * reply.setClientId(message.getReplyToclientId());
     * reply.setMessageContent(message.getMessageContent());
     * reply.setMessageDate(new Date());
     * reply.setMessageType(MessageType.ACCEPT_SECRET_KEY);
     * messages.put(message.getReplyToclientId(), reply);
     */
    String encryptedSecretKey = (String) message.getMessageContent();
    log.debug("encryptedSecretKey - " + encryptedSecretKey);
    String tempSecretKey = encryption.do_RSADecryptionKey(encryptedSecretKey, privateKey);
    this.myKey = encryption.getSecretKey(tempSecretKey);
    encryption.setSecretKey(this.myKey);

    log.debug("Secret key has been recevied and processed on the client" + clientId);

    Message secretMessage = new Message();
    secretMessage.setClientId(clientId);
    secretMessage.setMessageDate(new Date());
    secretMessage.setMessageType(MessageType.MESSAGE);
    secretMessage.setMessageContent("Secret key has been recevied");
    this.sendMessage(secretMessage);
```

Fig 13: Decryption of encrypted AES key (on client side) by using client's private key

In this code block we can see that the method defined in Figure 12 is called. This is on the client side – the AES key is received from the admin as encrypted text. The ciphertext is passed to the function, as well as the client's own private key – after which the AES key is decrypted and obtained as a string. The string/AES key is converted to a SecretKey object and set as the client symmetric key. This marks the end of the handshake and key exchange, and the client is now ready to encrypt and broadcast their messages to the entire group. Received messages from the group can also be decrypted with this key, so the client can read the messages.

6.2 Performance Evaluation

We want to evaluate the performance of the implementation of the key exchange process, with various increasing numbers of clients. This is important as speed is essential in cloud computing working environments – simply securing the environment should not be a time-intensive hassle.

The metrics are calculated in seconds, with a batch script timer and automated client setup: requesting for secret key, sending public key, receiving encrypted AES key, decrypting, and broadcasting a hello message. The time starts once the first client starts to generate their RSA key pair. The admin initialization is not included in the calculation as generating an AES-128 secret key for the group is of constant time O(1) and has insignificant time during other clients' initialization. See Figure 13 below for the graph of times in seconds for a group with an increasing number of clients joining.

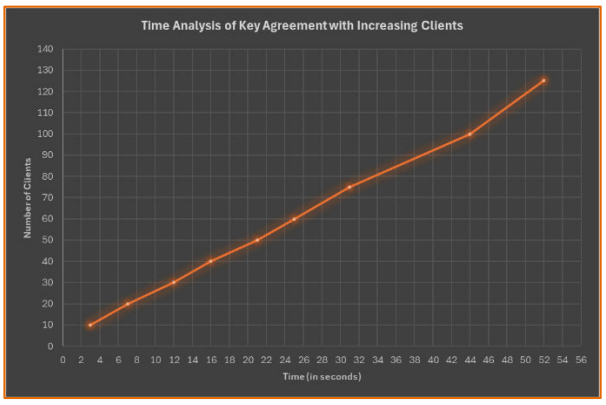

Fig 14: Time Analysis of Key Agreement with Increasing Clients

We can see that our implementation has a linear algorithmic complexity of O(n). As the number of clients increases, the graph steadily increases by an average factor of 5 seconds for an increase of ten clients – about 0.5s or 500 milliseconds for a single client to complete the handshake with the admin to obtain the secret key. It is to be noted that, for the purposes of analysing the time and complexity of our implementation, clients connected simultaneously, and the server application opened many threads to handle requests from each client – however, in a real world scenario where users would like to message each other in a group, client numbers do not exceed more than 200 in general, and do not

| ISSN: 2349-6037 | www.ijmserh.com ||Impact Factor: 9.274||Quarterly, Peer Reviewed & Refereed Journal|

|| Volume 13, Issue 2, April-June 2025 ||

DOI: 10.15662/IJMSERH.2025.1302037

connect at the same time – in addition, as stated before the handshake is completed in around 500ms or less, which is a negligible amount of time to ensure security when joining a group.

VII. CONCLUSION

In conclusion, we described our implementation example, i.e., a group messaging application on a cloud server; to demonstrate the use of symmetric key encryption in a cloud computing scenario. We described and defined the system architecture, the secure key sharing scheme, and necessary functions to enable encryption and decryption of the symmetric key with RSA public and private key pair.

The application was tested with multiple amounts of clients to analyse and record the time taken to complete the handshake and exchange of the secure symmetric key via an insecure cloud server – in proportion to increasing number of clients. We saw that the complexity of our implementation was O(n), or linear time. It was determined based on the collected data that on average, it took a client 500 milliseconds to generate their RSA public/private key pair, request for the group secret key by providing a public key, after receiving which the key was decrypted and client ready for the encryption and decryption of the group messages, for sending and receiving respectively.

REFERENCES

1. Jian Shen, Tianqi Zhou , Debiao He , Yuexin Zhang, Xingming Sun, and Yang Xiang IEEE TRANSACTIONS ON DEPENDABLE AND SECURE COMPUTING, VOL. 16, NO. 6, NOVEMBER/DECEMBER 2019
2. L. Zhou, V. Varadharajan, and M. Hitchens, “Cryptographic rolebased access control for secure cloud data storage systems,” IEEE Trans. Inf. Forensics Secur., vol. 10, no. 11, pp. 2381–2395, Nov. 2015.
3. F. Chen, T. Xiang, Y. Yang, and S. S. M. Chow, “Secure cloud storage meets with secure network coding,” in Proc. IEEE Conf. Comput. Commun., 2014, pp. 673–681.
4. L. Law, A. Menezes, M. Qu, J. Solinas, and S. Vanstone, “An efficient protocol for authenticated key agreement,” Des. Codes Cryptography, vol. 28, no. 2, pp. 119–134, 2010.
5. X. Yi, “Identity-based fault-tolerant conference key agreement,” IEEE Trans. Depend. Secure Comput., vol. 1, no. 3, pp. 170–178, Jul.– Sep. 2004
6. R. Barua, R. Dutta, and P. Sarkar, “Extending joux’s protocol to multi-party key agreement (extended abstract),” in Proc. 4th Int.Conf. Cryptology India, 2003, pp. 205–217
7. J. Shen, S. Moh, and I. Chung, “Identity based key agreement protocol employing a symmetric balanced incomplete block design,” J. Commun. Netw., vol. 14, no. 6, pp. 682–691, 2012
8. B. Dan and M. Franklin, “Identity-based encryption from the weil pairing,” SIAM J. Comput., vol. 32, no. 3, pp. 213–229, 2003.
9. S. Blakewilson, D. Johnson, and A. Menezes, “Key agreement protocols and their security analysis,” in Proc. IMA Int. Conf. Cryptography Coding, 1997, pp. 30–45
10. I. Chung and Y. Bae, “The design of an efficient load balancing algorithm employing block design,” J. Appl. Mathematics Comput., vol. 14, no. 1, pp. 343–351, 2004.
11. O. Lee, S. Yoo, B. Park, and I. Chung, “The design and analysis of an efficient load balancing algorithm employing the symmetric balanced incomplete block design,” Inf. Sci., vol. 176, no. 15, pp. 2148– 2160, 2006.
12. Daruvuri, R., Patibandla, K., & Mannem, P. (2024). Leveraging Unsupervised learning for workload balancing and resource utilization in cloud architectures. International Research Journal of Modernization in Engineering Technology and Science, 6(10), pp. 1776-1784.

INTERNATIONAL STANDARD SERIAL NUMBER INDIA

INNO SPACE SJIF Scientific Journal Impact Factor

doi cross ref

निस्केयर NISCAIR

International Journal of Multidisciplinary and Scientific Emerging Research (IJMSERH)

Impact Factor: 9.274

ijmserh@gmail.com

www.ijmserh.com