Scalability Analysis of Encryption Algorithms in Secure Group Messaging

Alexandru Lazarina, Vlad Manolescu, Amir Kalantarzadeh, Bati Gozen,
Karol Plandowski, Aukje Heijthuijzen, Vilmos Udvari
Group 3

June 10, 2025

Abstract

Secure group chat applications are essential for both personal and professional communication, with platforms such as WhatsApp, Signal, and Matrix relying on encryption to protect user privacy. End-to-end encryption (E2EE) ensures that only the intended recipients can access messages, preventing third parties from intercepting or tampering with data. However, as the number of participants in a group chat grows, encryption and key management become increasingly complex, leading to potential delays in message delivery, higher computational costs, and increased network load. Understanding how these encryption schemes scale in terms of performance is crucial for designing cryptographic protocols that maintain robust security while ensuring low latency and high scalability.

Contents

1 Introduction

The problem being addressed is understanding how various encryption schemes scale with the number of users in a group chat environment. Specifically, this study focuses on comparing the impact of multiple encryption algorithms, namely AES-GCM, Double Ratchet, TreeKEM (MLS), and Hybrid (AES + RSA/ECIES), on performance as group sizes increase. The main challenge is how encryption affects message latency, computational efficiency, and scalability within secure group messaging systems.

Despite the availability of different encryption protocols, there is no clear understanding of how they perform under increasing group sizes. While some protocols are known for their strong security guarantees, they may not scale efficiently, leading to delays or high computational costs.

Key management is a major challenge in group messaging. AES-GCM encrypts efficiently but lacks secure key distribution. Double Ratchet ensures strong forward secrecy but adds computational overhead. TreeKEM (MLS) optimizes rekeying for scalability, while Hybrid Encryption (AES + ECIES) balances AES speed with ECIES security for better security.

By benchmarking these encryption methods in a simulated group chat environment, this study aims to provide valuable insights into how encryption affects real-world group messaging performance.

The main questions driving this study are:

1. How does encryption overhead scale with the number of users in a group chat environment across different encryption algorithms (AES-GCM, Double Ratchet, TreeKEM, Hybrid AES+RSA/ECIES)?
2. Which encryption method provides the best balance of security, computational efficiency, and scalability for large-scale secure group messaging?
3. How do group size, input variables, and encryption algorithms impact key management efficiency, message propagation time, and overall system performance?

2 Methods and Implementation

This study aims to analyze the scalability of encryption in secure group chat environments by benchmarking and comparing different encryption schemes using different group sizes. The primary focus is on understanding how the encryption time, overhead, and key management complexity change as the number of users increases. The chosen encryption algorithms AES-GCM, Double Ratchet, TreeKEM, and Hybrid AES+RSA/ECIES are tested in a simulated environment where their impact on performance is measured. The encryption algorithms were selected based on their relevance in modern secure messaging systems and their differing approaches to balancing security, speed, and scalability.

2.1 AES-GCM

- **Reason for selection:** A symmetric encryption algorithm that secures digital communication by encrypting 128-bit blocks using 10, 12, or 14 rounds, depending on the key size. It operates on a 4x4 byte matrix and ensures security through Key Expansion, SubBytes, ShiftRows, and MixColumns. AES-GCM enhances security with GMAC (Galois Message Authentication Code), which ensures data integrity using Galois Field multiplication (GF(2)). GMAC generates an authentication tag, preventing tampering and replay attacks. Since GMAC is parallelizable, AES-GCM is much faster than CBC-MAC, making it ideal for secure protocols like TLS and IPsec.
- **Expected Behavior:** AES-GCM is expected to perform efficiently in small-group scenarios due to its high-speed encryption. However, in large-group environments, it may introduce considerable overhead because it requires encrypting a separate copy of the message for each recipient, resulting in scalability challenges.
- **Existing Uses:**
  - Secure Network Communications: TLS/SSL (HTTPS), VPNs, WI-FI Security (WPA2/WPA3)
  - End-to-end encryption in messaging and Secure Apps: Signal, WhatsApp, Telegram
  - Financial and Payment Security: EMV Chip Cards, Cryptocurrency Wallets

2.2 Double-ratchet

- **Reason for Selection:** Double Ratchet secures messages by using a combination of: (1) a symmetric-key ratchet that derives a new message key for every ciphertext by advancing an Hash-based Message Authentication Code (HMAC) chain, and (2) a Diffie-Hellman (DH) ratchet that refreshes the shared secret with new DH key pairs. Together they provide forward secrecy and post-compromise recovery.

  Our study compares two variations of the Double Ratchet algorithm:

1. **Pairwise Double Ratchet:** This is the original form used in Signal, where each pair of users maintains a separate Double Ratchet session. In a group of n users, this requires $O(n^2)$ sessions and significant encryption overhead as each message must be encrypted individually for every recipient. While secure, this model scales poorly to large groups. Real-world applications like WhatsApp improve efficiency using the *Sender Key* protocol, which limits the cost to one broadcast key per sender but reduces individual forward secrecy guarantees.
2. **Tree-based Group Double Ratchet:** To improve scalability, we implemented a group-capable variant using a binary hash tree structure inspired by TreeKEM. This design retains the core Double Ratchet mechanisms while enabling efficient group key updates and message broadcasts. Our implementation uses *sender keys* and a balanced *Binary hash tree*, achieving the following:
   (a) **Efficient group keying:** A binary hash tree of all member's keys produces a shared root secret only known to current members. When a user joins, leaves, or refreshes their DH key, the tree is partially updated, recalculating the root secret in logarithmic time. This new root then generates fresh sender keys for all members.
   (b) **Per-sender ratchet chains:** Each member derives a personal sending chain key from the root secret. For each message, the sender utilizes an HMAC-based derivation to obtain the encryption key from their chain, encrypting and broadcasting the ciphertext to all recipients. Since the root secret is regenerated on every membership change and leaf refresh, each sender's chain is reseeded, making any compromised keys ineffective after an update, thereby maintaining full forward secrecy.
   (c) **Message recovery:** Each encrypted message includes labels indicating the current epoch and the sender's chain index. These tags enable recipients to select the correct key and derive any skipped keys in case of messages arriving out-of-order.

- **Expected Behavior:** The algorithm provides secure, efficient messaging in small groups. However, as group size grows, overhead may increase. The use of sender keys mitigates this by limiting encryption overhead to a single broadcast key, while the binary hash tree facilitates efficient key management and preserves forward secrecy. keys.

- **Existing Use:** Double Ratchet is widely used in platforms like WhatsApp or Signal due to its strong security. WhatsApp addresses group messaging overhead by employing the sender key protocol, along with rotation of the sender keys, to maintain forward secrecy and lower the risk of compromise. Similarly, the IETF's upcoming Messaging Layer Security (MLS) standard uses a tree-based group key derivation approach on every membership change or update, ensuring that large groups also benefit from forward secrecy and post-compromise recovery.

2.3 TreeKEM

- **Reason for selection:** TreeKEM (Tree-based Key Encapsulation Mechanism) is designed specifically for efficient and scalable group messaging in dynamic environments where users frequently join or leave. Traditional group encryption schemes require generating and distributing a new key to every member when group membership changes, which becomes inefficient as group size grows. In contrast, TreeKEM organizes users into a binary tree structure, where each leaf node represents a user and internal nodes hold shared keys. When a user joins or leaves, only the path from the affected node to the root needs to be updated, significantly reducing the number of key operations and messages required.

- **Expected Behavior:** Improved scalability, as key updates affect only part of the group rather than the entire network. The protocol ensures forward secrecy and post-compromise security while maintaining efficiency during group evolution, making it ideal for secure group chats.

- **Existing Use:** While TreeKEM is not yet widely used in many messaging apps, it has been proposed as the next standard for large-scale encrypted communication, and it is very likely that it will be used in messaging apps in the future.

2.4 Hybrid (AES + RSA/ECIES)

- **Reason for Selection:** Hybrid encryption combines the efficiency of symmetric encryption (AES) with the security of asymmetric encryption (ECIES or RSA). This is especially useful in group chats, where encrypting each message for multiple recipients can be costly. Encrypting the message once with AES and distributing the AES key using ECIES or RSA balances performance and security.

- **Expected Behavior:** Each message gets a unique AES key for encryption. This key is then encrypted separately for each recipient using their ECIES or RSA public key. Encryption operations increase linearly with participants.

- **Existing Use:** Commonly used in secure communication protocols like TLS, which uses asymmetric encryption for key exchange and symmetric encryption for data. Secure email and cloud storage systems also rely on this approach to combine security with efficiency.

2.5 Environment

The study is conducted within a simulated group chat environment developed using a local Python-based testing framework. This setup enables controlled experimentation and performance evaluation of various encryption algorithms under different group sizes and message conditions. The simulation does not rely on any external network infrastructure, ensuring consistent timing and performance metrics.

The chat system was implemented with the following tools and technologies:

- Python
- Dash by Plotly : Used to create an interactive graphical user interface (GUI) for real-time visualization of encryption metrics such as encryption time, decryption time, and message overhead.
- Cryptographic Libraries: Python Crypto Library
- Resource Monitoring: Modules such as `psutil` and `time` were used to capture CPU usage, memory usage, and timing metrics.

All tests were performed on local machines with the following specifications:

- Experiment 1: MacBook M1 8gb RAM
- Experiment 2: Ryzen 5 3600 32gb RAM

3 Experiments

3.1 Encryption Overhead and Scalability

Research Question Addressed:
RQ1 - How does encryption overhead scale with the number of users across different encryption algorithms?
RQ2 - Which method provides the best performance–security trade-off for group communication?

Experimental setup:
To simulate group messaging scenarios under different communication loads, we varied group size and message size across all five encryption schemes. Each experiment was run **10 times per configuration** to ensure statistical significance and reduce variability due to outliers.

- **Group Sizes:** Simulated group chats with the following number of users:
  - 5 users (small group)
  - 50 users (medium group)
  - 100 users (large group)
  - 1,000 users (very large group)
  - 10,000 users (massive group)
- **Message Sizes:** To reflect different communication loads, the following message sizes were tested:
  - **Small:** 128 bytes (short messages)
  - **Medium:** 1024 bytes (typical text messages)
  - **Large:** 10240 bytes (media attachments or long messages)
- **Repetition and Averaging:**
  - For each combination of group size and message size, the encryption and decryption process was executed **10 times per algorithm**.
  - Metrics were averaged over these 10 runs to account for variability and obtain a robust representation of performance.

Metrics Collected:
Each encryption algorithm was evaluated using the following performance metrics:

- **Encryption Time (ms):** Time required to encrypt a message.
- **Decryption Time (ms):** Time required to decrypt a message.
- **CPU and Memory Usage (%):** Resource usage measured during encryption and decryption operations.
- **Total Time** : Total time to perform everything including setup of algorithms.

Objective: The goal of this experiment is to analyze the computational cost and performance impact of each encryption method as the group size and message size scale. The collected data provides insight into how suitable each encryption scheme is for different levels of group communication, particularly in terms of latency and scalability.

3.2 Key Management and Rekeying Overhead

Research question Addressed:
RQ3 - How do group size, input variables, and encryption algorithms impact key management efficiency, message propagation time, and overall system performance?

Experimental Setup:
This experiment focuses on the behavior of encryption algorithms when group membership changes, which is critical for secure communication. We simulate two key events:

- **User Join Event:** A new participant is added to the group, triggering a rekeying process.
- **User Leave Event:** An existing participant exits the group, also triggering a rekeying process.

Each event was tested across multiple group sizes (5, 50, 100, 1,000, and 10,000 users). Every scenario was run **5 times per algorithm** to obtain averaged rekeying performance metrics.

Metrics Collected:

- **Rekeying Time (ms):** Time required to recalculate and redistribute keys following a group membership change.

Objective: The purpose of this experiment is to evaluate how well each encryption algorithm manages dynamic group membership, particularly focusing on the computational cost and latency introduced by rekeying. Efficient key management is essential for maintaining security and performance in large-scale or frequently changing group communication scenarios.

4 Results

This section presents the comparative evaluation of five encryption schemes: AES-GCM, Double Ratchet, Hybrid (AES + RSA/ECIES), RatchetTree, and TreeKEM, under varying group sizes. Each algorithm was assessed based on **encryption time**, **decryption time**, and **total time**. These are the results averaging across all message sizes.

These are averaged times over 10 simulations per test. Key findings from these initial tests are as follows:

4.1 Total Time (Experiment 1)

Table 7 aggregates encryption and decryption along with any protocol overhead into a total time metric. This gives a complete view of overall communication latency.

Table 1: Total Time (ms) vs. Group Size

Group Size	AES-GCM	Double Ratchet	Hybrid	RatchetTree	TreeKEM
5	8.685	24.437	5082.895	8.636	1329.490
50	42.522	462.749	52203.854	36.499	1594.664
100	72.060	1725.919	105023.000	58.174	1700.545
1000	319.386	170032.414	1042228.080	289.429	3967.020
10000	2885.174	1700234.110	10595236.723	2622.022	27398.273

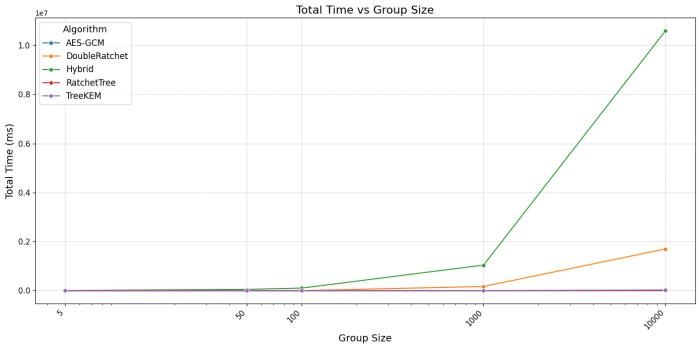

Figure 1: Total Time VS Group Size

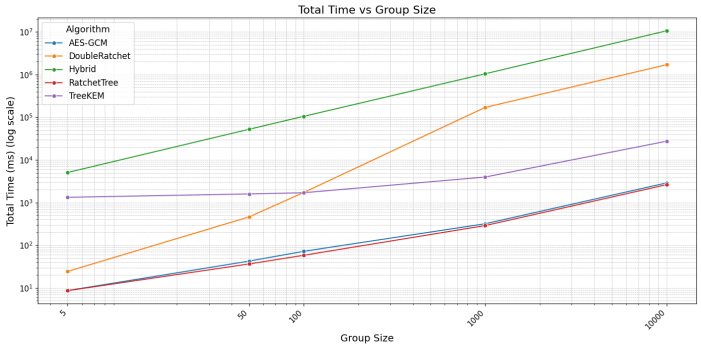

Figure 2: Total Time VS Group Size (log y scale)

- **Ratchet Tree and AES-GCM** are the most efficient overall for total time, with Ratchet Tree even outperforming AES-GCM at 10,000 users (2.6 seconds vs. 2.9 seconds).

- **Double Ratchet and Hybrid** schemes are the least scalable, with total times becoming prohibitively high for large groups due to their significant overhead. For 1,000 users, the total time for Double Ratchet was approximately 170 seconds.
- **TreeKEM** remains a viable option for large groups, with a total time of about 27.4 seconds for 10,000 users, which is significantly better than Double Ratchet and Hybrid methods.

4.2 Encryption Time

Table 2 shows the encryption times in milliseconds as group size increases.

Table 2: Encryption Time (ms) vs. Group Size

Group Size	AES-GCM	Double Ratchet	Hybrid	RatchetTree	TreeKEM
5	0.655	0.624	0.180	0.962	1.847
50	5.782	2.126	1.791	8.494	21.027
100	9.778	4.358	3.418	14.269	42.348
1000	48.432	44.737	33.614	67.560	426.066
10000	437.538	447.372	211.871	586.153	4267.906

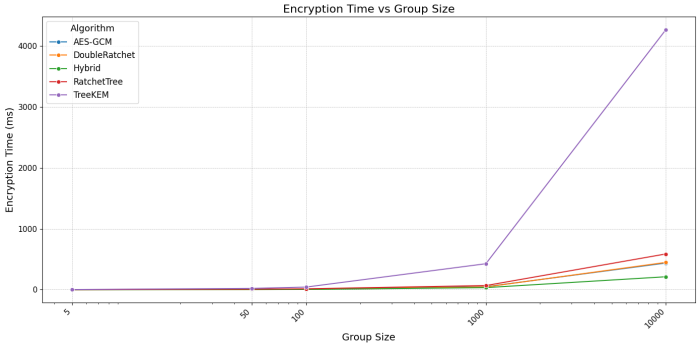

Figure 3: Encryption Time VS Group Size

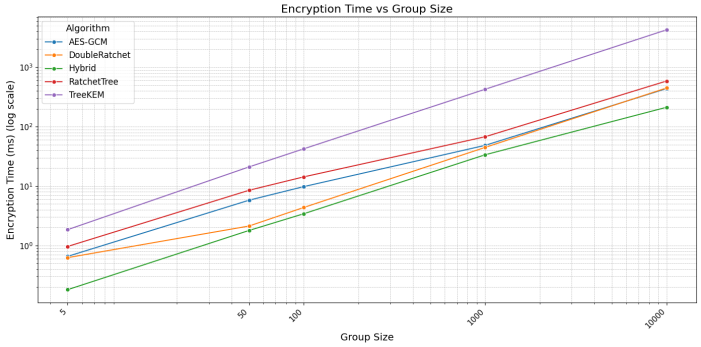

Figure 4: Encryption Time VS Group Size (log y scale)

- **AES-GCM** and **Ratchet Tree** show a relatively flat and gradual increase in encryption time, making them efficient for smaller to medium-sized groups.
- **TreeKEM** starts with a higher baseline encryption time for small groups but scales sub-linearly, meaning its encryption time grows slower than the group size, making it more suitable for very large groups.
- **Double Ratchet and Hybrid (AES + RSA/ECIES)** exhibit a steep increase in encryption time as the group grows. This is because they require operations that scale with the number of users. Double Ratchet involves separate encryptions for each pair of users, and the Hybrid method wraps the AES key for each recipient individually.

4.3 Decryption Time

Decryption times for the same scenarios are shown in Table 3.

Table 3: Decryption Time (ms) vs. Group Size

Group Size	AES-GCM	Double Ratchet	Hybrid	RatchetTree	TreeKEM
5	0.952	0.877	0.082	0.876	8.429
50	8.216	3.045	0.931	7.262	102.997
100	14.082	6.401	1.873	11.598	207.436
1000	68.082	65.772	18.932	60.288	2083.943
10000	615.245	657.725	79.820	611.198	20942.517

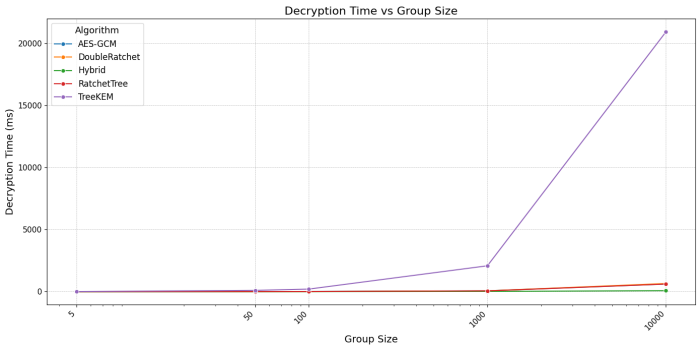

Figure 5: Decryption Time VS Group Size

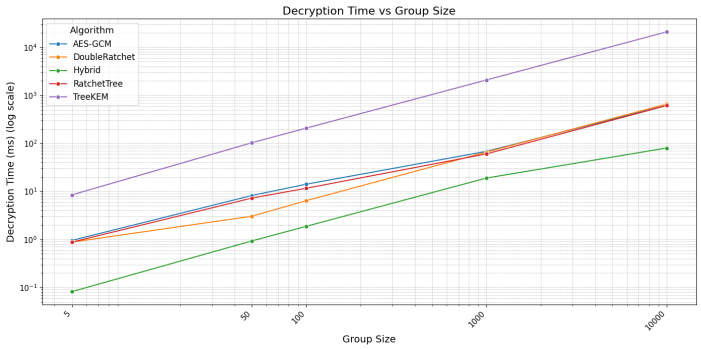

Figure 6: Decryption Time VS Group Size (log y scale)

- **Hybrid** shows the fastest decryption times, with only 79.820 ms for a 10,000-user group.
- **AES-GCM, Double Ratchet, and Ratchet Tree** have comparable decryption times that increase with group size.
- **TreeKEM** has the longest decryption times, which increase significantly with group size, reaching over 20 seconds for 10,000 users.

4.4 CPU/Memory Usage

This subsection presents the CPU and memory usage incurred during encryption and decryption processes across varying group sizes.

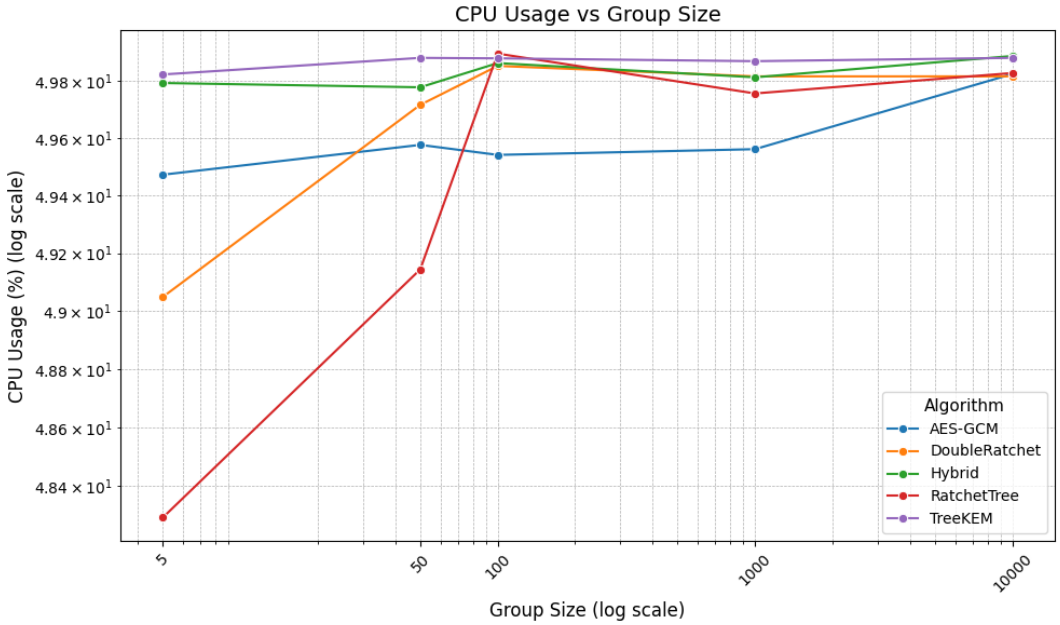

Figure 7: CPU Usage VS Group Size

- **AES-GCM and Ratchet Tree** are shown to consume the least amount of CPU.
- **TreeKEM** has a higher CPU cost per message but low overhead for rekeying.
- **Double Ratchet and Hybrid** methods exhibit the highest overall CPU load in large group scenarios.

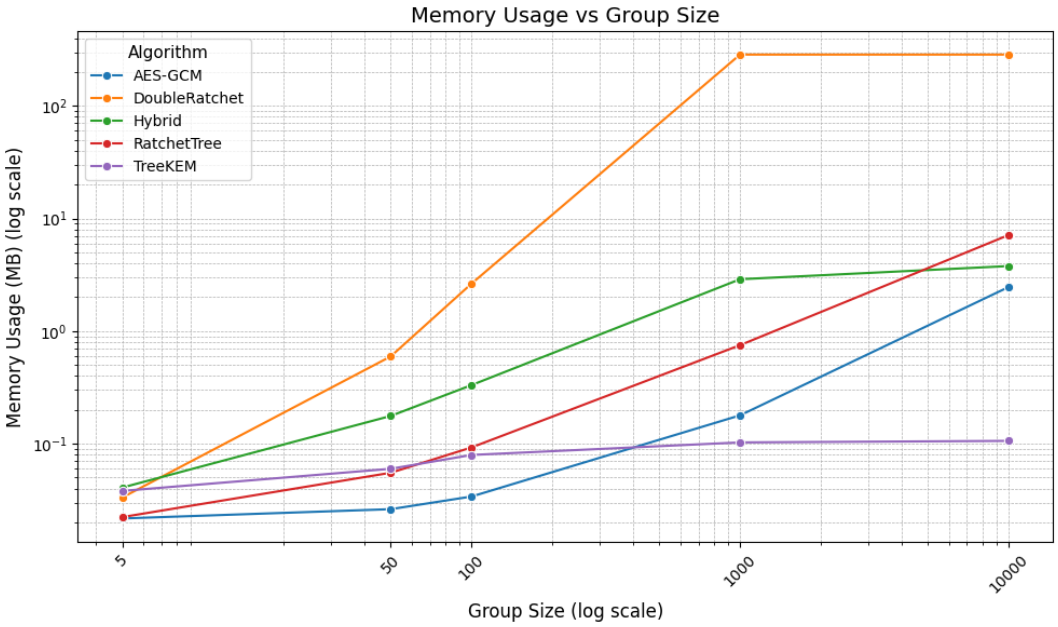

Figure 8: Memory Usage VS Group Size

- **AES-GCM and TreeKEM** are the most memory-efficient.
- **Double Ratchet and Ratchet Tree** show a significant increase in memory consumption, likely due to the need to maintain state for each pair of users.
- **Hybrid** also shows a steady increase in memory usage as the group grows.

4.5 Time VS Message Size

As message sizes grow, the time required to process them can increase due to the computational demands of the encryption algorithm and its associated operations. This section investigates how each algorithm scales in terms of encryption and decryption time as the plaintext size increases.

Table 4: Encryption and Decryption Time (ms) per Algorithm and Message Size

Encryption Time

Algorithm	128 B	1 KB	10 KB
AES-GCM	27.33	52.75	220.70
DoubleRatchet	29.70	53.44	216.40
Hybrid	46.31	47.50	56.71
RatchetTree	58.59	84.77	263.11
TreeKEM	951.69	951.47	952.35

Decryption Time

Algorithm	128 B	1 KB	10 KB
AES-GCM	67.43	93.88	261.30
DoubleRatchet	74.43	105.36	260.50
Hybrid	16.66	18.22	26.10
RatchetTree	67.73	90.61	256.40
TreeKEM	4668.46	4658.87	4679.87

Table 5: Total Time (ms) per Algorithm and Message Size

Algorithm	128 B	1 KB	10 KB
AES-GCM	497.89	575.22	913.80
DoubleRatchet	373510.25	374869.70	375161.77
Hybrid	2363738.96	2359698.42	2356427.35
RatchetTree	525.37	556.29	727.20
TreeKEM	7160.53	7138.05	7295.42

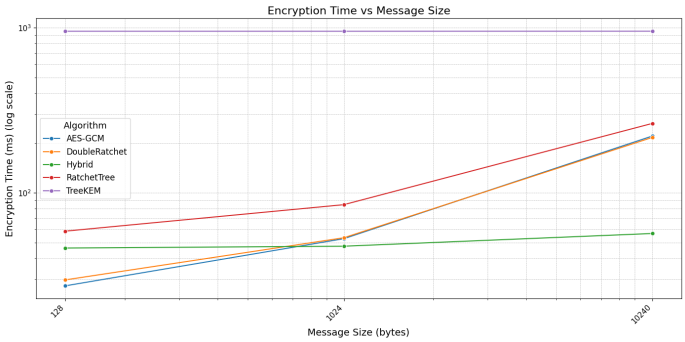

Figure 9: Encryption Time vs Message Size

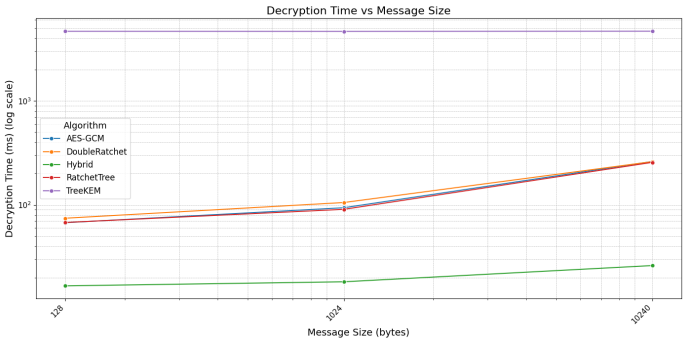

Figure 10: Decryption Time vs Message Size

- **TreeKEM's** performance is largely independent of the message size. The time it takes to encrypt or decrypt does not change significantly whether the message is 128 bytes or 10 KB. This suggests its overhead is related to key management rather than the data volume.
- All other algorithms (**AES-GCM, Double Ratchet, Ratchet Tree, and Hybrid**) show a strong, direct correlation between message size and processing time. As the message gets larger, the time to encrypt and decrypt increases.

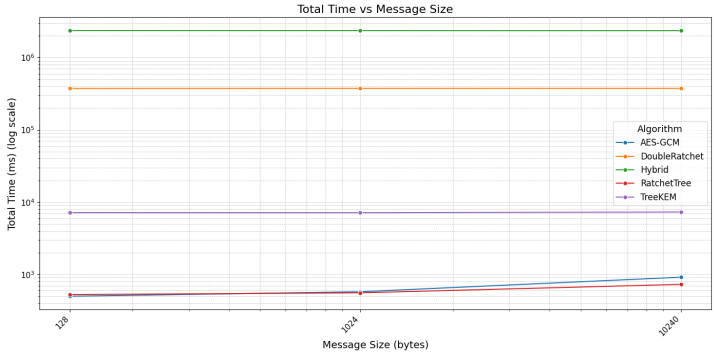

Figure 11: Total Time Vs Message Size

- The total time follows a similar pattern, with TreeKEM showing little change with message size, while the other algorithms' total time increases as messages get larger.

4.6 Rekeying Time (Experiment 2)

Efficient rekeying ensures that forward and backward secrecy are maintained without incurring significant performance penalties. This subsection analyzes the time required to perform rekeying operations for each encryption algorithm across different group sizes.

4.6.1 Simulating User Join Event

Table 6: Rekeying Time for **Add** Operation (in ms)

Group Size	AES-GCM	Double Ratchet	Hybrid	RatchetTree	TreeKEM
5	0.212	1.483	788.797	0.400	480.619
50	3.758	11.889	778.432	1.850	822.907
100	15.023	24.667	1259.828	3.685	965.897
1000	1679.268	833.554	2046.916	33.347	1284.680
10000	197954.267	8866.843	8125.167	347.287	2030.368

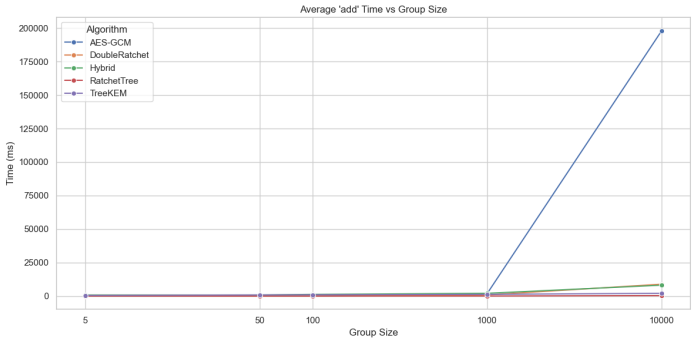

Figure 12: User ADD time comparison

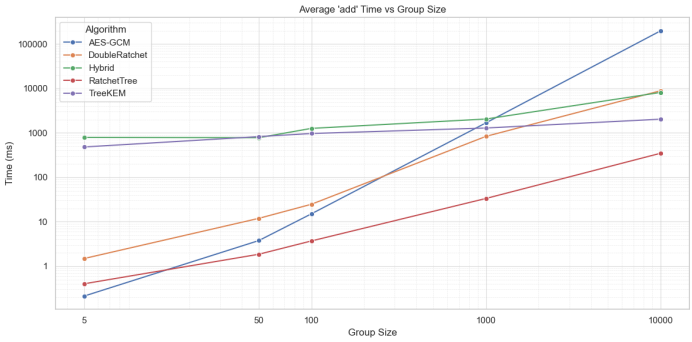

Figure 13: User ADD time comparison (log y scale)

- **Ratchet Tree** is the most efficient for adding users in groups up to 1,000 members.
- **TreeKEM and Ratchet Tree** demonstrate the best scalability and become the fastest for very large groups (10,000 users). Tree-based structures like these are designed for efficient user additions.
- **Double Ratchet and Hybrid** algorithms scale poorly when adding users.
- **AES-GCM's** rekeying time for adding a user dramatically increases for very large groups, reaching almost 198 seconds for 10,000 users.

4.6.2 Simulating User Leave Event

Table 7: Rekeying Time for **Remove** Operation (in ms)

Group Size	AES-GCM	Double Ratchet	Hybrid	RatchetTree	TreeKEM
5	0.177	0.109	4.457	0.329	483.972
50	3.542	0.352	32.530	1.732	800.373
100	14.486	0.869	67.063	4.678	915.160
1000	1714.288	86.691	612.565	46.294	1308.393
10000	197576.981	127.546	6315.993	334.370	1771.379

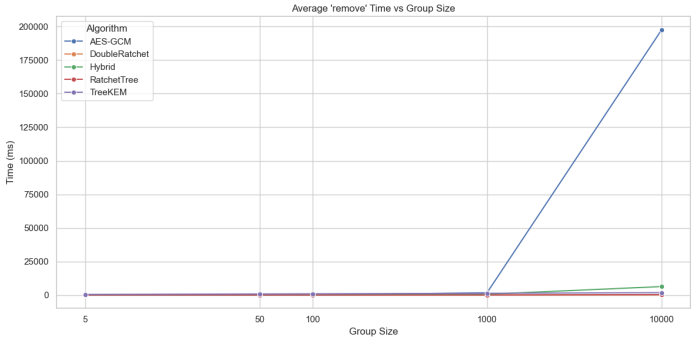

Figure 14: User REMOVE time comparison

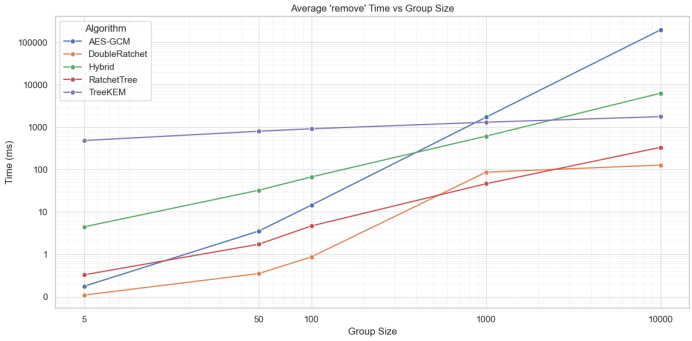

Figure 15: User REMOVE time comparison (log y scale)

- **Pairwise Double Ratchet** is exceptionally fast at removing users, significantly outperforming all other methods across all group sizes.
- **AES-GCM** again shows poor scalability for user removal in very large groups.
- **Ratchet Tree and Hybrid** methods also show efficient user removal times when compared to AES, with Hybrid being on the slower end.
- **TreeKEM** has a higher overhead for user removal compared to other methods, though it remains relatively stable across group sizes.

4.7 Statistical analysis

4.7.1 Hypothesis I: AES-GCM is the most time efficient algorithm overall

Statistical Analysis of Time Efficiency:

A one-way analysis of variance (ANOVA) was performed to assess differences in overall time efficiency among four cryptographic algorithms: AES-GCM, Hybrid, RatchetTree, and TreeKEM. The analysis incorporated both encryption/decryption times and the average of rekeying times (add and remove operations) for each algorithm, evaluated across five group sizes (5, 50, 100, 1000, and 10,000). For each algorithm, ten time measurements (five for encryption/decryption and five for rekeying) were included, resulting in a total of forty data points.

The ANOVA compared the mean overall time required by each algorithm. The results showed no statistically significant differences in time efficiency between the algorithms, with an **F-value of 1.2572** and a **p-value of 0.3036 ($F(3, 36) = 1.2572$, $p = 0.3036$)**. This indicates that, under the tested conditions and when considering both encryption/decryption and rekeying operations together, none of the algorithms demonstrated a statistically significant advantage or disadvantage in terms of time efficiency.

Interpretation:

Although AES-GCM is often considered for its efficiency, the statistical analysis did not find evidence to support that it is significantly faster or slower than Hybrid, RatchetTree, or TreeKEM when both encryption/decryption and rekeying times are taken into account across the tested group sizes. This suggests that, for the scenarios evaluated, the choice among these algorithms may be guided by factors other than overall time efficiency.

4.7.2 Hypothesis II: The tree-based Double Ratchet (RatchetTree) offers the best overall rekeying performance and scalability

Statistical Analysis of Rekeying Efficiency:

To evaluate the performance of rekeying operations, a separate statistical analysis was conducted on the four relevant algorithms: the pairwise DoubleRatchet, the tree-based RatchetTree, Hybrid, and TreeKEM. Due to the small sample size ($n = 5$) for each test condition, a non-parametric Kruskal-Wallis H-test was performed to compare the algorithms at each group size (5, 50, 100, 1,000, and 10,000) for both 'add' and 'remove' operations. This was followed by a Dunn's post-hoc test with Bonferroni correction to identify specific pairwise differences.

The Kruskal-Wallis tests were statistically significant ($p < 0.001$) across all tested conditions, confirming substantial performance differences among the algorithms. The post-hoc analysis revealed a clear trade-off between the algorithms rather than a single best performer.

Key Statistical Findings:

- **'Add' Operation Scalability:** For adding new members, RatchetTree was the best performing algorithm up to 10,000 users. TreeKEM was fast, though its performance was not statistically different from RatchetTree. Both tree-based methods were significantly faster and more scalable than the DoubleRatchet and Hybrid algorithms, which scaled poorly.
- **'Remove' Operation Performance:** For removing members, the pairwise DoubleRatchet was exceptionally efficient, significantly outperforming all other algorithms at all group sizes. However, it was inconsistent for some of the values. If we look at the graphs, they showed us for scalability, the best option is TreeKEM again.
- **RatchetTree vs. Pairwise DoubleRatchet:** The analysis highlights a critical design trade-off. The RatchetTree significantly improves scalability for adding users compared to the original DoubleRatchet but at the cost of performance for removing users.

Interpretation:

The hypothesis that the RatchetTree offers the best overall performance is not fully supported, as the optimal algorithm is dependent on the specific operation and group size. The statistical evidence shows that tree-based structures (RatchetTree and TreeKEM) are essential for scalable user additions, aligning with the theoretical O(log N) complexity. The pairwise DoubleRatchet, while extremely fast for removals, is not a scalable solution for growing groups due to its O(N) overhead for other operations. Therefore, the choice of a rekeying algorithm for a real-world system involves a critical trade-off: TreeKEM and RatchetTree are superior for large, dynamic groups where users join frequently, while the original DoubleRatchet would be preferable only if efficient user removal is the absolute priority.

4.7.3 Hypothesis III: The encryption and decryption time for TreeKEM is largely independent of message size, whereas the performance of all other tested algorithms shows a significant positive correlation with message size.

Statistical Analysis of Performance:

To investigate the impact of message size on performance, a statistical analysis was conducted to test the hypothesis that the encryption and decryption times for TreeKEM are largely independent of message size, while the performance of other algorithms is directly correlated with it. The analysis employed Pearson's correlation coefficient (r) to quantify the linear relationship between message size (128, 1024, and 10240 bytes) and the corresponding processing times for each algorithm.

The results confirm that TreeKEM's performance is statistically independent of the message size. For both encryption and decryption, the correlation coefficient was negligible and the relationship was not statistically significant (**encryption $r = 0.015$, $p > 0.99$**; **decryption $r = 0.165$, $p = 0.90$**). This indicates that the performance cost of TreeKEM is **not driven by the volume of data being processed.**

In stark contrast, all other tested algorithms exhibited a **strong, positive, and statistically significant correlation** between processing time and message size. Algorithms including AES-GCM, DoubleRatchet, and Ratchet Tree all showed correlation coefficients (r) **exceeding 0.99 ($p < 0.05$)** for encryption time. A similar strong positive correlation was observed in their decryption times.

Interpretation:

The analysis validates the initial hypothesis, demonstrating a fundamental architectural difference in TreeKEM. Its performance overhead is primarily associated with key-management operations rather than the per-byte processing of the message itself, unlike the other symmetric and hybrid methods evaluated.

5 Discussion

The experimental results present a nuanced landscape where the optimal encryption strategy is not universal but is instead highly dependent on the specific context of the group chat environment, particularly its size and membership dynamism.

The statistical analysis of overall time efficiency, which combined encryption, decryption, and rekeying operations, showed no single algorithm to be significantly superior, with an **ANOVA F-value of 1.2572** and a **p-value of 0.3036**. This lack of a statistically significant winner indicates that the choice between these protocols must be guided by factors beyond a simple aggregate time measurement, such as the specific operational requirements and scalability demands of the application.

A primary takeaway is the critical trade-off between the immediate performance of symmetric encryption and the long-term scalability of more complex, tree-based key management structures.

- **AES-GCM**, while unequivocally the fastest for static or low-churn groups, lacks an inherent scalable key distribution mechanism.
- This limitation becomes apparent as group sizes grow, where its overhead increases considerably.
- Also, protocols designed explicitly for scalability, like **TreeKEM** and the tree-based Double Ratchet, demonstrate their strength in larger groups.
- Their **logarithmic complexity** for user additions is a clear advantage over the poor scaling of the Pairwise Double Ratchet and Hybrid algorithms for the same operation.

Furthermore, the analysis of rekeying efficiency reveals a crucial design conflict. The statistical tests for rekeying operations were significant **($p < 0.001$)**, highlighting substantial performance differences.

- The **Pairwise Double Ratchet** was exceptionally efficient at removing users.
- However, this strength is overshadowed by its poor scalability for adding users, making it an impractical choice for growing communities.

This leads to a critical trade-off: **TreeKEM and Ratchet Tree** are superior for large, dynamic groups where users join frequently. The choice of a rekeying algorithm therefore involves a critical decision between optimizing for **user joins** or **removals**, with the former being the priority for scalable systems.

These findings allow for the formulation of a clear decision-making framework for developers of secure messaging platforms:

- For small-scale or static groups, the raw speed of AES-GCM makes it the most time-efficient choice.
- For medium-sized groups, the tree-based Double Ratchet (Ratchet Tree) offers a pragmatic middle ground, providing latency close to AES-GCM while drastically cutting rekeying time for membership changes.
- For large and very large dynamic groups, TreeKEM emerges as the scalability leader. It is the most effective solution when strong forward secrecy must be preserved across thousands of users with frequent membership churn, despite a higher per-message latency.
- The Pairwise Double Ratchet and Hybrid (AES+RSA/ECIES) schemes, due to their significant performance and overhead costs in larger scenarios, are only appropriate in niche applications where their specific security properties outweigh the substantial performance penalties.

6 Conclusion

This study measured the performance of five end-to-end–encryption strategies: AES-GCM, Double Ratchet, RatchetTree, TreeKEM and a Hybrid AES+RSA/ECIES, across group sizes from 5 to 10,000 members. The key findings are shown below.

Answering our first research question, encryption latency rose with group size for every scheme, but at markedly different rates. AES-GCM kept the flattest curve (0.7 ms -> 0.44 s). RatchetTree shadowed it closely up to 1 000 users and, in total-time terms, even edged it at 10 000 (2.6 s vs 2.9 s). TreeKEM began with higher cost (1.8 ms) yet increased sub-linearly, reaching $\approx$ 4.3s at 10 000 users. Double Ratchet and the Hybrid scheme climbed steeply, the former because every user pair encrypts separately, the latter because each AES session key is individually wrapped with RSA/ECIES.

For the second question, the "best" protocol depends on the balance between speed, security guarantees and membership churn:

- **Static or low-churn groups**: *AES-GCM* is unequivocally the fastest.
- **Very large, dynamic groups**: *TreeKEM* offers strong forward secrecy and the lowest rekey cost once membership changes ($\approx$ 2 s at 10 k users), at the expense of higher per-message latency.
- **Medium-sized groups**: *RatchetTree* provides a pragmatic middle ground—latency within 10 % of AES-GCM while cutting rekey time by three orders of magnitude ($\approx$ 0.3 s vs $\approx$ 198 s at 10 k users).
- **Double Ratchet**: Best reserved for $\leq$ 100 users; beyond that its pairwise overhead dominates (total time $\approx$ 170 s at 1 000 users).
- **Hybrid**: Layered security came with the largest cost envelope and offered no clear advantage in our threat model.

Regarding the final question, system load mirrored the latency trends. AES-GCM and RatchetTree consumed the least CPU and memory. TreeKEM imposed higher per-message CPU but minimal rekey overhead. Double Ratchet and Hybrid displayed the heaviest aggregate load in massive-group scenarios.

In short, *AES-GCM is the speed champion for small or static groups, RatchetTree is the balanced choice for mid-scale chats, and TreeKEM is the scalability leader when forward secrecy must be preserved in groups of thousands.* Double Ratchet and Hybrid are appropriate only when their specific security properties outweigh substantial performance penalties.

References

[1] K. Bhargavan, R. Barnes, and E. Rescorla, "Treekem: Asynchronous decentralized key management for large dynamic groups—a protocol proposal for messaging layer security (mls)," 2018, unpublished manuscript.

[2] K. Klein *et al.*, "Keep the dirt: Tainted treekem, adaptively and actively secure continuous group key agreement," in *IEEE Symposium on Security and Privacy (SP)*, 2021.

[3] B. Hamouda, "Comparative study of different cryptographic algorithms," *Journal of Information Security*, vol. 11, pp. 138–148, 2020.

[4] S. Surendran, A. Nassef, and B. D. Beheshti, "A survey of cryptographic algorithms for iot devices," in *2018 Long Island Systems, Applications and Technology Conference (LISAT)*, 2018.

[5] A. Ghosh, "Comparison of encryption algorithms: Aes, blowfish and twofish for security of wireless networks," *ResearchGate*, 2020.

[6] Y. Shin, J. Hur, and H. Yoon, "Scalable and efficient approach for secure group communication," in *International Symposium on Communications and Information Technologies*, 2009.

[7] Signal Messenger LLC, "Double ratchet," https://signal.org/docs/specifications/doubleratchet/, 2016, accessed: 2025-02-23.

[8] D. Collins, D. Riepel, and S. A. O. Tran, "On the tight security of the double ratchet," in *Proceedings of the 2024 ACM SIGSAC Conference on Computer and Communications Security (CCS '24)*, 2024.

[9] A. Bienstock, "A more complete analysis of the signal double ratchet algorithm," in *Advances in Cryptology – CRYPTO 2022*, 2022.

[10] M. Bellare, A. C. Singh, J. Jaeger, M. Nyayapati, and I. Stepanovs, "Ratcheted encryption and key exchange: The security of messaging," in *Advances in Cryptology – CRYPTO 2017, Part III*, J. Katz and H. Shacham, Eds., vol. 10403. Springer, Cham, 2017, pp. 619–650.

[11] M. Mozaffari-Kermani and A. Reyhani-Masoleh, "Efficient and high-performance parallel hardware architectures for the aes-gcm," *IEEE Transactions on Computers*, vol. 61, no. 8, pp. 1165–1178, 2012.