JCBI
Journal of Computing & Biomedical Informatics
ISSN: 2710 - 1606
Volume 11 Issue 02
2026

Research Article
https://doi.org/10.56979/1102/2026/1575

Lightweight Quantum-Resistant Group Key Management for Large-Scale IoT Networks Using Lattice Cryptography

Eimad Abusham[1], Ahmad Al Kayed[1], Abdelrahman H. Hussein[2], Sagheer Abbas[3*], and Taher M. Ghazal[1,2,4]

[1]Faculty of Computing and IT, Sohar University, Oman.
[2]Department of Networks and Cybersecurity, Hourani Center for Applied Scientific Research, Al-Ahliyya Amman University, Amman, Jordan.
[3]Department of Computer Science, Prince Mohammad Bin Fahd University, Alkhobar, 31952, KSA.
[4]Center for Cyber Security, Faculty of Information Science and Technology, Universiti Kebangsaan Malaysia (UKM), 43600 Bangi, Selangor, Malaysia.
[*]Corresponding Author: Sagheer Abbas. Email: jamsagheer@gmail.com

Received: May 20, 2026 **Accepted:** August 28, 2026

Abstract: Secure multi-recipient Internet of Things communication in resource-constrained environments requires cryptographic protocols that are light computational requirements, scalable in bandwidth for large group sizes, and quantum-resistant. Existing group and dual-receiver encryption schemes achieve efficiency using elliptic curve cryptography but lack post-quantum security, which could be compromised by polynomial-time computations on powerful quantum computers. In contrast, current lattice-based alternatives incur prohibitive signature overhead and are unsuitable for embedded deployment. Hence, the research introduces LAGDR (Lattice-Authenticated Group Keying with Dynamic Rekeying), a protocol combining ML-KEM-512 key encapsulation for quantum-resistant key agreement, TinyJAMBU authenticated encryption for lightweight bulk data protection, and Falcon-512 signatures-the first consistent 128-bit post-quantum framework integrating ultra-lightweight AEAD with a fast-verifying compact lattice-based signature scheme for scalable multi-party IoT communication scenarios. Unlike Conventional per-recipient encryption paradigms, LAGDR employs a binary tree key hierarchy, achieving O(log n) rekeying complexity and reducing 512-member group operations from 512 encryptions to 9 while maintaining 666-byte signatures and 800-byte public keys. TinyJAMBU's hardware-optimized permutation (150 cycles/byte)for bulk encryption and Falcon's NTRU structure for sub-5ms signature verification on ARM Cortex-M4 microcontrollers. ML-KEM 512's reduced key sizes enhance bandwidth efficiency without compromising quantum security for target IoT applications.

Keywords: Multi-Recipient Encryption; Lattice Cryptography; Group Key Management; Internet Of Things (IoT) Security; Dynamic Rekeying; Post-Quantum Signatures

1. Introduction

Large-scale quantum computing represents an existential threat to the current public-key cryptographic infrastructure [1]. Recent developments in lightweight cryptography have addressed some of these issues with elliptic curve-based encryption schemes, enabling higher computational efficiency and smaller key sizes [2]. In [3], a quantum blockchain framework incorporating lattice-based signatures is impractical for many embedded applications, even though signature sizes exceed 2 KB. . Point-to-point encryption further complicates this when communicating with multiple recipients. Group communication systems need to accommodate dynamic membership, facilitate effective key distribution and rekeying, and provide forward and backward secrecy as group members enter or exit [4, 6]. Public-key cryptosystems are considered traditional and are not applicable in resource-constrained environments where

computational power, memory, and energy are severely limited [5]. In [6], ecur dynamic resource allocation and secure information management in blockchain-based systems, specifically computation overhead in cryptographic processes. The growth of the Internet of Things (IoT) in vehicular networks, also known as the Internet of Vehicles (IoV), has presented unparalleled challenges in ensuring multi-recipient communication [7], especially when constrained devices are involved, which impose prohibitive computational requirements and thus are not suitable for large-scale deployment in power- and resource-limited systems [8].

Traditional methods that unicast messages to different people scale readily, but their computational and communication requirements increase linearly with group size [9]. However, these traditional cryptographic solutions face a looming threat from quantum computing, which will pose an insurmountable challenge to today's public-key infrastructure via Shor's algorithm [10]. Quantum adversaries will be able to use Shor's algorithm to efficiently solve the integer factorization problem and the discrete logarithm problem, rendering RSA and elliptic curve cryptography (ECC) completely insecure [11]. The key issue with post-quantum group communications is the trilemma of quantum resistance, lightweight resource usage, and logarithmic scalability, which is a three-way optimization challenge. Current schemes based on broadcasting encrypt the message copy for each recipient; this incurs a communication complexity of $O(n)$, which is not feasible for groups with more than n hundred members [12]. Nevertheless, these implementations are usually not combined with post-quantum key establishment protocols and with scalable group management protocols [13].

1.1. Main Contribution

We propose a new cryptographic model, LAGDR (Lattice-Authenticated Group Keying with Dynamic Rekeying), by integrating ML-KEM-512, a quantum-resistant key encapsulation mechanism; tinyJAMBU, an ultra-lightweight authenticated encryption mechanism; and Falcon-512, a fast-verifying digital signature mechanism, into the first consistent 128-bit post-quantum security architecture optimized for minimal resources to support multi-party communication in the Internet of Things. The protocol's binary-tree key hierarchy reduces rekeying complexity from $O(n)$ to $O(log\ n)$, enabling scalable group administration in networks with hundreds of participants while keeping cryptographic operations practical on microcontroller-like devices. The work addresses the urgent need for practically deployable quantum-resistant group communication systems that are resource-intensive enough to meet the high resource demands of contemporary IoT settings. Therefore, the main contributions can be listed as follows:

- A unified lightweight post-quantum cryptographic architecture tailored for multi-recipient IoT communication.
- Efficient integration of lattice-based primitives with ultra-lightweight encryption to minimize latency and resource consumption.
- A dynamic rekeying mechanism enabling secure and scalable group communication.
- Comprehensive analysis demonstrating improved performance-security trade-offs compared to existing approaches.

1.2. Outline

The remainder of this paper is organized as follows. Section 2 surveys related work in post-quantum cryptography and multi-recipient protocols. Section 3 formalizes system models, primitive specifications, and security definitions. Section 4 details the LAGDR protocol's algorithms, provides a complexity analysis, and presents security proofs. Section 5 evaluates performance on ARM Cortex-M4 hardware and provides a comparative analysis with existing benchmark schemes. Section 6 discusses the trade-offs and limitations of the proposed LAGDR protocol and outlines future directions to address them.

2. Literature Survey

Singamaneni et al. [14] recommended a hybrid quantum-crypto standard, based on lattice-based cryptography with quantum key distribution. 6G-enabled IoT NTRU encryption with 256-bit keys was implemented on ESP32 microcontrollers, using synthetic IoT traffic datasets. The scheme achieved a mean latency of 47 ms and 94.3% attack success against quantum-foe attacks. However, the linear n-overhead scaling (O(n)) in group communication requires single-device encryption, which is impractical for large broadcast operations involving more than 100 nodes in a bandwidth-constrained environment. Peelam et

al. [15] improved by using NIST-qualified FALCON-512 signatures and lattice-based encryption. They tested the solutions on SUMO-generated vehicular mobility traces of 500 vehicles in an urban topology of 10 km 2. The study reported a 12.3 ms signature creation time and a 2.1 ms verification time, with 99.1%. Research Gap: . The signatures-per-vehicle scheme incurs the 500x bandwidth overhead of dynamically broadcasting messages over constant bandwidth schemes, which have not been scaled to more than 1000 vehicles in dense urban deployments.

Qiao et al. [16] construct CCA-secure IBE-KEM with a persistent leaks (1- ε) fraction of secret key revelation in dual-system encryption on groups of composite order, and demonstrated in the standard model without random oracles based on Waters' dual-system approach. The 128-bit security parameter was experimentally tested: encapsulating a key took 89 ms, and the resulting ciphertext was 156 KB. Significant Weakness: PERCIP requires O(n) computation and O(n) ciphertext transmission, offering no support for efficient one-to-many broadcast encryption, which is needed for IoT command broadcasting.

2.1. Cloud Pairing-Free Multi-Recipient Searchable Encryption

Shaw and Dutta [17] proposed lattice-based multi-user signcryption, which is existential unforgeable and IND-CCA2-secure, based on the NTRU construction with key privacy, and constructs 2,048-bit NTRU parameters without specifying the dataset using by-definition security proofs. : The paper reports 78 ms of signcryption for a single recipient and an overhead of 1.8 KB. The scheme involves an independent signcryption operation per participant, resulting in O (n) multiplicative cost, and has not been empirically evaluated on real IoT devices or real-world deployments to validate its scalability.

Abdmeziem et al. [18] addressed asynchronicity in group keys by deriving keys using time-based key derivation trees, which they simulated in the Cooja emulator and tested with 50 Contiki-OS sensor nodes under intermittent connectivity. This indicated an average rekeying latency of 180 ms (97.2%) and a key delivery success rate. Linear rekeying costs O(n) to change the members; not post-quantum adversarial cryptographic primitives; not supporting constant-overhead broadcast encryption, member addition/removal incurs total group rekeying overheads, which in turn create disparities between cloud-based PHR sharing and traditional in-house scenarios. Sukte et al. [19] used the Improved Shark Smell Optimization (ISSO) algorithm to automatically generate keys for cloud-based PHR sharing using a 300-patient diabetes dataset. Key optimization efficiency and time to encrypt were 94.6% and 23 ms per record, respectively. Weaknesses: ISSO heuristic optimization lacks theoretical security proofs; O(n)-per-patient encryption prevents scalable hospital-wide publicity; no quantum-resistance analysis; single-sender, no multi-recipient optimization for simultaneous patient-group publicity.

Wang et al. [20] proposed ORCA, a revocable group covert channel on the blockchain with steganographic transaction embedding and distributed key agreement, and tested it on the Ethereum mainnet with 50 participants sending 1,000 covert messages each. . We achieved 89.3% undetectability and 1.80 seconds of message latency at a logarithmic revocation cost of O(log n). Blockchain-based infrastructure overheads are not feasible for lightweight IoT; stenography interest is misplaced given authenticated broadcast encryption, and no quantum-resistant primitives exist that support real-time operation with little-to-no latency control, which is vital for industrial IoT controller systems.

Yang et al. [21] constructed multi-recipient searchable keyword encryption using bilinear pairings, but without them, using a Trapdoor mechanism based on ElGamal. They experimented on a synthetic document database with 10,000 encrypted files and 50 recipients simultaneously. Obtained 156 ms per encrypted message per recipient and 0.95 search accuracy with a 0.89 recall. Linear recipient overhead (n156 ms) makes the scheme impractical for broadcast with n over 100; it is not post-quantum-secure, making it susceptible to the Shor algorithm and thus not quantum-safe, a terminal requirement for future-proof IoT deployment.

Xu and others [22] proposed lightweight multi-message multi-receiver signcryption with elliptic curve cryptography (ECC-256) and batch verification, which they tested in a vehicle network simulated with NS-3, with 200 vehicles sending traffic-safety messages. It was found that 8.7 ms batch signcryption with 10 messages yielded 89.4% bandwidth savings compared to single-message encryption. . Quantum attacks target the classical ECC foundation. Currently, batching signcryption on a blockchain with n recipients and m messages has O(n - m) complexity, without optimizing overhead by constant factors. It does not provide hierarchical key management to support scalable group dynamics. Yang et al. [23]

evaluated multi-message signcryption on top of a blockchain with tamper-proof communication, with n recipients and O (n - m) message complexity, without constant-factor indirectness optimization or hierarchical key management. They achieved an average time of 234 ms for signcryption with blockchain-verifiable integrity, providing 99.7 per cent deterrence against tampering. Consensus delays, currently 1.2 seconds per message, constitute blockchain overhead; the O(n) complexity of recipients remains; no quantum-resistant primitives exist; and the need for blockchain infrastructure, incompatible with resource-constrained offline IoT devices with sub-100 ms response times, persists.

2.2. Research Gap Identification

The computation or communication complexity for each scheme surveyed is O(n), with n representing the number of recipients [14-23]. The average time required for each recipient to operate is between 12 and 234 ms [15,17,23]. Bandwidth does not exceed n×0.8-1.8 KB per recipient and scales linearly [14,17,21]. All of the schemes surveyed require O(n) operations for the entire group size to achieve a constant O(1) broadcast overhead. This limitation makes them unsuitable for large-scale IoT deployments where cumulative latency exceeds 50-100 ms with fewer than 500 devices, and exceeds the average bandwidth available in 2G or 3G cellular networks (40-80 Kbps) with more than 500 devices.

Group Key Management currently in use today [18,20] requires rekeying all members when adding or removing members, resulting in O(n) rekeying operations per member addition or removal. Additionally, experiments by [18] show that 50 nodes take 180 ms to rekey, and [20] uses logarithmic rekeying for member revocation, but at the cost of higher latency due to blockchain use (1.8 seconds). None of the existing methods can simultaneously achieve (1) forward secrecy, (2) post-compromise security, (3) logarithmic O(log n) rekeying complexity, and (4) <200 ms of total time required to rekey for groups with >500 members, which is necessary to recover quickly from security breaches (to achieve responsive security). Although [14-16] include quantum-resistant primitives (NTRU, FALCON, lattice IBE), they do not provide low-resource hardware, i.e., an ARM Cortex-M4 with 128 KB RAM and 168 MHz CPU, but instead rely on desktop computations [16,18] or high-end,d unaffordable microcontrollers [14]. ECC/pairing-based classical schemes [21-23] can still be defeated by Shor's algorithm, without providing a way to migrate to post-quantum schemes. This is because the literature does not describe NIST-defined PQC (ML-KEM-512, Falcon-512) and hierarchical key management that achieve sub-10 ms encryption and sub-2 ms decryption on Cortex-M4.

3. Research Methodology

3.1. System Model

The system model provides a network system design and cryptographic primitives with concrete parameter instantiations, a hierarchical key structure, and security requirements for the LAGDR formal model.

Let Group $G = \{P_1, P_2, \dots, P_n\}$ of n numbers having a key distribution center (KDC) on the server S with an adversary A with quantum capabilities. Each has an ML-KEM-512 key pair (pk_i) operating on resource-constrained devices such as ARM Cortex-M4, 128 KB RAM, 168 MHz clock, and a sender entity encrypting messages for the group using the current root key K_{root} without requiring individual member keys. The KDC handles group membership and distributes keys, but it is not authorized to learn any message content; it only facilitates authentication and key agreement. Communication between KDCs and groups authenticated broadcast, unless communication between two parties (e.g., initial registration) is required, in which case point-to-point secure channels are used in practice over cellular networks (2G/3G/LTE with supporting 40-384 Kbps uplink) or LoRaWAN infrastructure. A probabilistic polynomial-time quantum adversary A with capabilities: 1) passive eavesdropping on all broadcasts, 2) active message modification detected using signatures, 3) corruption of $t < n$ members learning secret keys $\{sk_j\}$, and 4) quantum computation executing Shor's algorithm. Security goals need IND-CCA2 confidentiality with an advantage $Adv^{IND-CCA2}(A) \leq 2^{-\lambda}$ for $\lambda = 128$, integrity detecting tampering with probability $1 - 2^{-\lambda}$ forward secrecy, ensuring past messages are secure after compromise, and post-compromise security, ensuring future messages are secure after rekeying.

3.2. Cryptographic Primitive Specifications

The ML-KEM-512 module, armed with an error key encapsulation mechanism, is employed for key establishment, and the scheme operates over the polynomial ring $R_q = Z_q[X]/(X^n + 1)$ with parameters $(n, k, q) = (512,2,3329)$. The key features follow the error distribution centered binomial B_η with $\eta = 2$; each coefficient is $a - b$ for independent uniform bits $a, b \in \{0,1\}$. The compressed matrix $A \in R_q^{k \times k}$ and vector in Equations (1) and (2) as follows.

$$t = As + e \; ; \; s \leftarrow^{\$} B_\eta^k \; e \leftarrow^{\$} B_\eta^k \quad (1)$$

$$(c_1, c_2) = (A^T r + e_1, t^T r + e_2 + Encode(m)) \quad (2)$$

provides 800 bytes; ciphertext size: 768 bytes; components $c_1 = A^\top r + e_1$ and $c_2 = t^\top r + e_2 + Encode(m)$. The shared secret key is 256-bit $ss = SHAKE - 256(m \parallel ct)$, derived using the Fujisaki-Okamoto transform, ensuring IND-CCA2 security under the Module-LWE assumption. Breaking ML-KEM-512 requires solving an LWE instance of dimension $d = k \cdot n = 1024$ with BKZ block size $\beta \approx 410$, demanding $\approx 2^{138}$ quantum operations.

TinyJAMBU-128 authenticated encryption with associated data scheme selected from NIST lightweight cryptography employs a compact 288-bit state $(S_0 \parallel S_1 \parallel S_2)$ split into three 96-bit registers processed through a non-linear feedback shift register with an update function $S_i' = S_{i-1} \oplus ((\neg S_{i-2}) \wedge S_{i-3}) \oplus K[i mod 128]$, combining the previous state and scheduled cyclically.

Digital signature scheme based on the NTRU lattice problem over $R_q = Z_{12289}[X]/(X^{512} + 1)$ polynomial ring with public key $h = g \cdot f^{-1} \in R_q$(897 bytes), secret key as (g) sampled from a discrete Gaussian with $\sigma = 1.17\sqrt{q}$ stored as an LDL decomposition of the Gram matrix, enabling fast FFO sampling and signing message 1281 bytes for fast sampling. The product short vector $s = (s_1, s_2)$ satisfying $s_1 h - s_2 = c(mod q)$ where $c = HashToPoint(m)$ is derived using rejection sampling from D_σ $(\sigma = 1.55\sqrt{q})$ discrete Gaussian, consuming 15. ms,2 ms for initial generation, compressing s_1, s_2 satisfies the norm bound $\| s_1 \|^2 + \| s_2 \|^2 \leq \beta^2$.

3.3. Hierarchical Binary Tree Key Structure

By organizing group keys in a binary tree T of height $h = \lceil log_2 n \rceil$ where n numbers occupy leaf positions, and intermediate nodes are stored in internal nodes, enabling $O(log\ n)$ scalability. Each node (l, i) at level $l \in [0, h]$ with position $i \in [0, 2^l - 1]$ maintains key $K_{l,i} \in \{0,1\}^{256}$, where level $l = 0$ denotes the root storing group master key $K_{root} = K_{0,0}$ and level $l = h$ contains leaves with member-specific keys $K_{h,i}$ for member P_i. For parent $K_{l,j}$ with context string left versus right and position (l, j) the key derivation is calculated as given in Equatiosn (3) and (4),

$$K_{l+1,2j} = HKDF - Expand(K_{l,j}, "left" \parallel l \parallel j \parallel \epsilon), \text{for left child} \quad (3)$$

$$K_{l+1,2j+1} = HKDF - Expand(K_{l,j}, "right" \parallel l \parallel j \parallel \epsilon), for\ right\ child \quad (4)$$

These formulations ensure domain separation, preventing key correlation attacks across tree branches, and epoch ϵ provides temporal separation across rekeying events.

3.4. Security Definitions

Definition 1: Multi-Recipient Encryption

IND-CCA2 security for the multi-recipient encryption scheme $\Pi = (Setup, Encrypt, Decrypt)$ achieves indistinguishability under adaptive chosen-ciphertext attack if, for a probabilistic polynomial-time quantum adversary A, the advantage is negligible in the security parameter λ, where the experiment $Exp_{\Pi,A}^{IND-CCA2}(1^\lambda)$ proceed as,

1) Challenger runs $\left(\{(pk_i, sk_i)\}_{i=1}^n, K_{root}\right) \leftarrow^{Setup} (1^\lambda, n)$ generating keypairs for n members and a shared root key, providing public keys $\{pk_i\}$ and server public keys pk_S to the adversary while keeping the secret keys and the root key private, as defined in Equation (5).

$$Adv_\Pi^{IND-CCA2}(A) = | \, Pr\,[Exp_{\Pi,A}^{IND-CCA2}(1^\lambda) = 1] - 1/2 \, | \leq 2^{-\lambda} \quad (5)$$

2) :The adversary makes a polynomial number of decryption oracle queries (i) for arbitrary ciphertext ct and member index $i \in [1, n]$ receiving plaintext $m = Decrypt(sk_i, K_{root}, ct)$ and rejection symbol $\perp$ if decryption fails.

3) Adversary submits a challenge consisting of two equal-length messages (m_1^*) and the recipient set $S^* \subseteq \{1, \ldots, n\}$, challenger samples random bit $b \leftarrow^{\$} \{0,1\}$ and returns challenge ciphertext $ct^* = Encrypt(K_{root}, m_b^*, S^*)$ encrypted for the recipient set S^*.

4) :The adversary continues making decryption queries (i) with the restriction that queries (i) for any $i \in S^*$ are forbidden.

5)Adversary outputs a guess $b' \in \{0,1\}$ attempting to determine which message was encrypted; the experiment returns 1 if $b' = b$ (correct guess), else 0.

In this definition, , we expand the traditional IND-CCA2 definition to accommodate multi-recipient environments, where each ciphertext may be delivered to multiple recipients, while preventing an adversary from distinguishing, for any two encrypted messages (ciphertexts) meant for different users/recipients, which user each is intended for. The adversary can access decryption oracles to simulate an adaptive chosen-ciphertext attack and observe the decryption of any chosen invalid ciphertext (invalid ciphertexts may cause error messages or timing differences between valid/invalid ciphertexts to leak information).

Definition 2: Forward Secrecy

A group encryption scheme provides forward secrecy if the compromise of the current epoch key $K_{\epsilon,t}$ at time t, it yields a negligible advantage in decrypting past ciphertexts for any PPT adversary A receiving the compromise $K_{\epsilon,t}$ and observes past ciphertexts $\{ct_{\epsilon,s}\}_{s<t}$ encrypted under previous epoch keys $\{K_{\epsilon,s}\}_{s<t}$, the probability $Pr\,[A(K_{\epsilon,t}, \{ct_{\epsilon,s}\}_{s<t}, public) \rightarrow m_i] \leq 2^{-\lambda}$ of recovering any past message m_i encrypted at Time $i < t$ remains negligible, where public includes all broadcast values, public keys, and ciphertexts. This property ensures historical communications remain confidential following future key compromise, critical for long-lived IoT deployments where devices may be compromised through physical capture, preventing the adversary from retroactively decrypting archived traffic recorded during the eavesdropping phase before compromise

Definition 3: Post-Compromise Security

A group encryption scheme achieves post-compromise security if, after detecting compromise and executing member removal, a new key $K'_{\epsilon+1} \leftarrow^{RemoveMember} (j, K_{\epsilon,t})$ for a compromised member $P_j$at Time t, future messages encrypted under regenerated key satisfy $Pr\,[A(sk_j, K_{\epsilon,t}, ct^{K'}_{\epsilon+1}) \rightarrow m] \leq 2^{-\lambda}$ in which the adversary possesses a compromised secret key sk_j and the old epoch key $K_{\epsilon,t}$ but cannot decrypt new ciphertexts $ct^{K'}_{\epsilon+1} = Encrypt(K'_{\epsilon+1}, m)$ with non-negligible probability. This feature guarantees that after a compromise is detected and mitigated, the system can restore security through rekeying, allowing it to recover rather than disbanding an entire group and rebuilding. LAGDR achieves PCS through fresh entropy injection during member removal, ensuring $K'_{\epsilon+1}$computationally independent of $K_{\epsilon,t}$despite a deterministic hierarchical structure.

4. LADGR Protocol Design

The following section outlines the complete LAGDR protocol specification in the form of five algorithms for the determination of initial keys for group setup, message encryption to be used by broadcast communication, message decryption by individual receivers, addition of a new member to the group by dynamic joining, and removal of a member by revocation.

Figure 1 illustrates the LAGDR framework and its dynamic, post-quantum-secure group communications. A trusted key distribution server manages dynamic group membership; individual secret keys are established using the ML-KEM-512 key establishment method. Each group member is authenticated by using Falcon-512 signature updates to ensure the integrity of rekeying updates against injection and replay attacks through unsecured communication channels (e.g., cellular networks, LoRaWAN). TinyJAMBU-128 authenticated encryption provides confidentiality for the sender, offers lightweight authenticated encryption for resource-constrained IoT nodes, and allows group members to independently verify the signature and generate new group keys to maintain confidentiality in the event of partial compromise of group keys (t < n). The system explicitly models both passive eavesdropping and an attacker with quantum-type capabilities to demonstrate its security against IND-CCA2 and EUF-CMA threats; it also supports scalability and incremental rekeying.

Problem Formulation

Design a multi-recipient encryption protocol that minimizes communication and computational costs while satisfying security and resource constraints. Formally, minimizing objective function $Cost = \alpha \cdot | Bandwidth | + \beta \cdot Latency + \gamma \cdot Memory$ where $\alpha, \beta, \gamma > 0$ represents deployment specific weights by

prioritizing bandwidth minimization with large α, real-time control prioritizes latency with large β, embedded systems prioritize memory with large α, real-time control prioritizes latency with large β, embedded systems prioritize memory with large γ subject to constraints:1) security achieving IND-CCA2, forward secrecy, and post compromise security with advantage $\leq 2^{-128}$ 2) Quantum Resistance: Minimum 128-bit post quantum security against adversary with quantum computer executing algorithm that requires from NIST PQC standardization, 3) Memory Constraint: Peak RAM Usage $\leq 100KB$ fitting IoT microcontrollers like ARM-Cortex-M4, 4) Latency: Per message encryption and decryption latency $\leq$ 20 ms enabling real-time control applications requiring sub-50 ms total processing, 5) Bandwidth Constraint: Per messasge overhead $\leq$ 3KB fitting single IP packet avoiding fragmentation over cellular networks with 1200-1500 byte MTU, 6) Scalability: Rekeying complexity $O(log\ n)$ enables efficient membership changes for large groups $n > 100$.

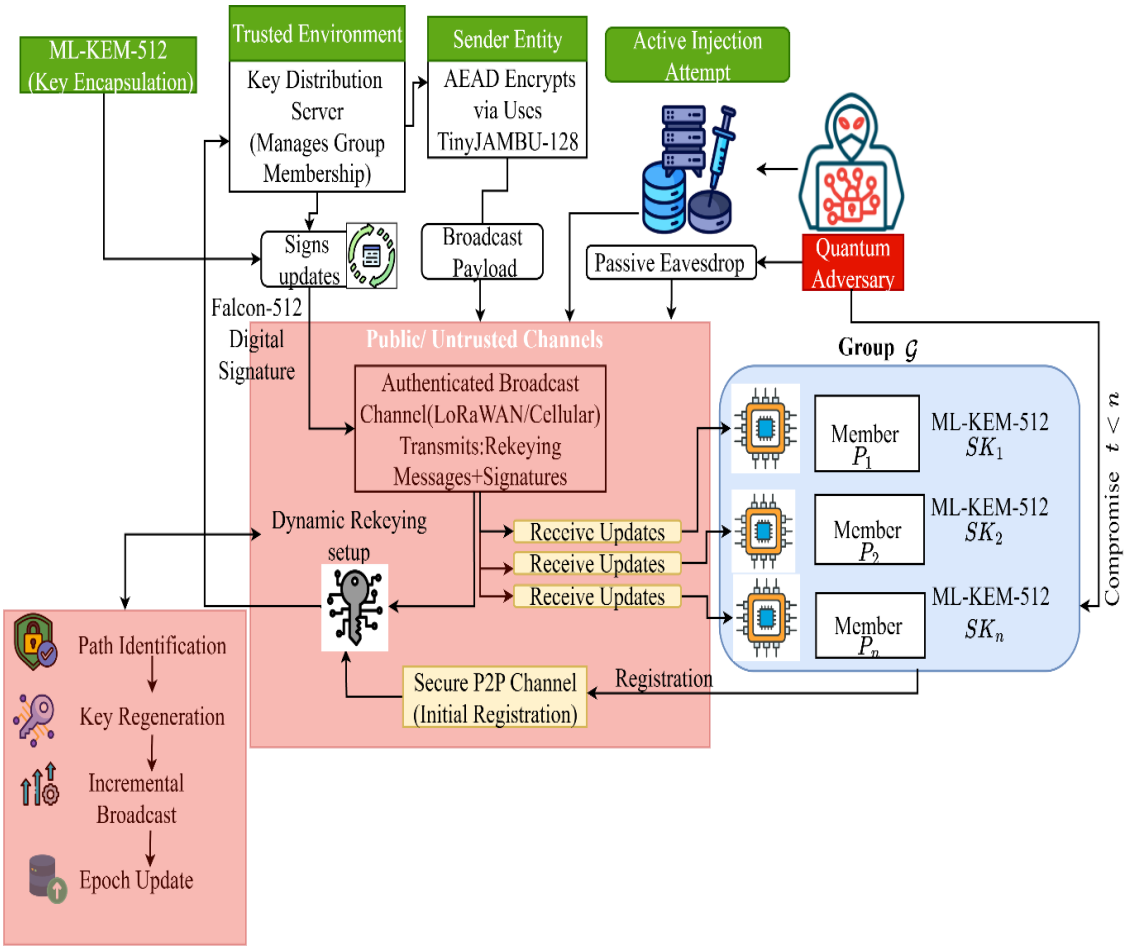


Figure 1. Overall Illustration of End-to-end architecture of the proposed LAGDR framework.

LAGDR solves this constrained optimization through: selecting ML-KEM-512 reducing public keys and ciphertexts while maintaining equivalent 128-bit quantum security, employing TinyJAMBU-128 achieving 150 cycles/byte (1.12 MB/s throughput on 168 MHz Cortex-M4) versus AES-128-GCM requiring 180 cycles/byte and larger state, adopting Falcon-512 generating 666-byte signatures versus SPHINCS+-128f producing 17,088 bytes (96% reduction) enabling sub-3KB overhead, organizing keys in binary tree achieving $O(log\ n)$storage per member (320 bytes for $n = 512$) and $O(log\ n)$rekeying operations (10 operations for $n = 512$), deriving child keys via HKDF ensuring deterministic reconstruction without storing entire tree, and rotating epochs via fresh entropy providing forward secrecy and post-compromise security through computational independence across temporal boundaries.

4.1. Group Setup Algorithm

The group setup establishes a shared root key called K_{root} among n no.of members using individual ML-KEM encapsulations integrated using an XOR combiner, builds a hierarchical binary tree structure of

height $h = \lceil log_2 n \rceil$, and authenticates setup parameters via Falcon signature, enabling verification of legitimate group configuration.

i) Server & Member Initialization: The server S generates a long-term Falcon keypair, publishing pk_S to all potential group members, while each member P_i generates an ML-KEM key pair and registers the public keys pk_i with the server as given in Equations (6) and (7).

$$(sk_S, pk_S) \leftarrow^{Falcon-512.KeyGen} (1^\lambda) \quad (6)$$

$$(sk_i, pk_i) \leftarrow^{ML-KEM-512.KeyGen} (1^\lambda) \quad (7)$$

ii) Encapsulation & Key Generation: For each member P_i server executes encapsulation, generating a 768-byte ciphertext ct_i and a 256-bit shared secret ss_i, accumulating total $n \times 768$ bytes ciphertext overhead is accumulated as given in Equation (8),

$$(ct_i, ss_i) \leftarrow^{ML-KEM-512.Encaps} (pk_i) \quad (8)$$

iii) XOR Combination of Shared Secrets: The server combines all shared secret keys $ss_{combined}$ through XOR, which offers theoretical secrecy where revealing fewer than n secrets provides zero information about the combined value $ss_{combined} = ss_1 \oplus ss_2 \oplus \cdots \oplus ss_n$.

Algorithm 1: $GroupSetup(1^\lambda, n, \{pk_i\}_{i=1}^n)$

Require: $\lambda \in N, n \in N, \{pk_i\}_{i=1}^n$

Ensure: $K_{root} \in \{0,1\}^{256}, T = \{K_{l,j}\}, \sigma_{setup}, Setup = \{ct_1, \dots, ct_n, \epsilon, gid, salt, \sigma_{setup}\}$

1. Initialize $(sk_S, pk_S) \leftarrow Falcon - 512.KeyGen(1^\lambda)$
2. $\varepsilon \leftarrow 0, gid \leftarrow UUID(), salt \leftarrow \${0,1}^{256}$
3. $SS \leftarrow \emptyset,\ CT \leftarrow \emptyset$
4. $\forall i \in \{1, \dots, n\}: (ct_i, ss_i) \leftarrow ML - KEM - 512.Encaps(pk_i)$
5. $CT \leftarrow CT \cup \{ct_i\}, SS \leftarrow SS \cup \{ss_i\}$
6. $combined \leftarrow \bigoplus_{i=1}^{n} ss_i$

$$IKM \leftarrow combined \,||\, \epsilon \,||\, gid$$

$$K_{root} \leftarrow HKDF - Extract(salt, IKM)$$

$$h \leftarrow \lceil log_2 n \rceil, T \leftarrow \emptyset, K_{0,0} \leftarrow K_{root}$$

$$\forall l \in \{0, \dots, h-1\}, \forall j \in \{0, \dots, 2^l - 1\}:$$

$$ctx_L \leftarrow "left" \parallel l \parallel j \parallel \varepsilon$$
$$K_{l+1,2j} \leftarrow HKDF - Expand(K_{l,j}, ctx_L, 256)$$
$$ctx_R \leftarrow "right" \parallel l \parallel j \parallel \varepsilon$$
$$K_{l+1,2j+1} \leftarrow HKDF - Expand(K_{l,j}, ctx_R, 256)$$
$$T \leftarrow T \cup \{K_{l+1,2j}, K_{l+1,2j+1}\}$$

end for

$$H_{msg} \leftarrow SHA - 256(msg)$$

$$\sigma_{setup} \leftarrow Falcon - 512.Sign(sk_S, H_{msg})$$

$$Setup \leftarrow \{ct_1, \dots, ct_n, \varepsilon, gid, salt, \sigma_{setup}\}$$

$$Broadcast(Setup)$$

$$return\ (K_{root}, T, \sigma_{setup})$$

Complexity:

$$Time: O(n \cdot T_{encaps} + 2^{h+1} \cdot T_{HKDF} + T_{sign}) Space: O(2^{h+1})$$

iv) Root Key Generation: Deriving the root key K_{root} and context, including epoch counter $\epsilon = 0$ and unique group identifier, ensuring different groups and epochs yield independent keys.

$K_{root} = HKDF - Extract(salt, combined \parallel \epsilon \parallel group_id)$ using fresh random $salt \leftarrow^{\$} \{0,1\}^{256}$.

v)Hierarchical Key Derivation: The server constructs a binary tree T by setting the root $K_{0,0} = K_{root}$ and recursively deriving children $K_{l+1,2j}$ and $K_{l+1,2j+1}$ for all levels $l \in [0, h-1]$ and positions $j \in [0, 2^l - 1]$ generating a complete tree with $2^{h+1} - 1$ total nodes required as given in Equations (9) and (10).

$\{K_{l+1,2j} = HKDF - Expand(K_{l,j}, "left" \parallel l \parallel j \parallel \epsilon, 256), K_{l+1,2j+1} = HKDF - Expand(K_{l,j}, "right" \parallel l \parallel j \parallel \epsilon, 256)$. (9)

$(2^{h+1} - 1) \times 0.004 = (2^{\lceil log_2 n \rceil + 1} - 1) \times 0.004ms$ (10)

Concretely, the tree contains $2^{h+1} - 1$ nodes, $(2^{10} - 1) \times 0.004 = 4.092ms$ for $n = 512 (h = 9)$ members on an ARM Cortex-M4.

vi) Signature Verification & Broadcast: The server authenticates the setup parameters by signing σ_{setup} using SHA-256 hash H, consuming 8.1 ms with tree caching FFT tables, and broadcasts totaling as given in Equations (11-13).

$M_{setup} = \{ct_1, \dots, ct_n, \epsilon, groupID, salt, \sigma_{setup}\}$ (11)

$\{\sigma_{setup} = Falcon - 512.Sign(sk_S, H(ct_1 \parallel ct_2 \parallel \cdots \parallel ct_n \parallel \epsilon \parallel group_id \parallel salt)) n \times 768 + 8 + 32 + 32 + 666 = \times 768 + 738\, 512 \times 768 + 738 = 393{,}954 bytes, where\ n = 512 \approx 384.7KB$ (12)

For the member-side processing, each member P_i independently verifies by consuming 1.0ms, and decapsulates ss_i recovering their shared secret key in 0.38ms in Equations (13) and (14).

$Falcon - 512.Verify(pk_S, H(\cdots), \sigma_{setup}) =? accept$ (13)

$ss_i = ML - KEM - 512.Decaps(sk_i, ct_i)$ (14)

Verifies the combined secret using published commitments $H(ss_j)$ and derives K_{root} locally matching the server computation, and derives path keys $\{K_{l,j} : (l,j) \in Path(i)\}$ requiring $h + 1 = 10$ HKDF expand operations totalling $10 \times 0.004 = 0.04$ ms, and per-member cost remains logarithmic $O(log\ n)$ after receiving the broadcast.

4.2 Multi-Recipient Message Encryption Algorithm

The LAGDR message encryption algorithm focuses on securely encrypting and broadcasting messages to multiple recipients. This process starts with the generation of an ephemeral message key $K_m \leftarrow^{\$} \{0,1\}^{256}$ for each broadcast, ensuring distinct keys across messages to prevent cross-message correlation attacks. The message key the message under TinyJAMBU authenticated encryption.

i) Message Key Generation: A 256-bit message key is generated using a hardware random number generator (TRNG) $K_m \leftarrow^{\$} \{0,1\}^{256}$ implemented using ring oscillator jitter and validated through continuous monitoring of NIST SP 800-90B entropy requirements, ensuring a minimum of 256 bits of entropy per sample.

Nonce Construction: The sender constructs a unique nonce $N = \epsilon \parallel timestamp \parallel counter$ totaling 128 bits, providing temporal and sequential uniqueness, 32-bit epoch identifier $\epsilon \in [0, 2^{32} - 1]$ incremented upon each rekeying event, providing coarse temporal separation across key rotation boundaries. The collision probability for q messages is bounded by $Pr[collision] \leq q^2 / 2^{129}$ which remains negligible even for extremely high message rates.

Algorithm 2: $MultiRecipientEncrypt(m, K_{root}, \varepsilon, gid, sender_id, sk_S)$

Require: $m \in \{0,1\}^*, K_{root} \in \{0,1\}^{256}, \varepsilon \in N, gid \in \{0,1\}^{128}, sender_id \in \{0,1\}^{64}, sk_S$

Ensure: $CT = (C, \tau, w_{root}, N, AD, \sigma), |\ CT\ | = |\ m\ | + 746$bytes

1. $K_m \leftarrow \${0,1}^{256}$
2. $timestamp \leftarrow GetCurrentTime()$
3. $counter \leftarrow GetMessageCounter()$
4. $N \leftarrow \varepsilon \parallel timestamp \parallel counter$
5. $AD \leftarrow sender_id \parallel \varepsilon \parallel gid$
6. $State \leftarrow TinyJAMBU - Init(K_m, N)$
7. $State \leftarrow TinyJAMBU - ProcessAD(State, AD)$

8. $n_{blocks} \leftarrow \lceil |\ m\ |/96 \rceil$
9. $C \leftarrow \varepsilon$
10. $\forall i \in \{1, \ldots, n_{blocks}\}$:

$$m_i \leftarrow m[(i-1) \cdot 96 : i \cdot 96]$$
$$State \leftarrow TinyJAMBU - Update^{640}(State)$$
$$keystream_i \leftarrow Extract(State, 96)$$
$$C_i \leftarrow m_i \oplus keystream_i$$
$$C \leftarrow C \parallel C_i$$

11. $State \leftarrow TinyJAMBU - Finalize^{1024}(State)$
12. $\tau \leftarrow Extract(State, 128)$
13. $N' \leftarrow \varepsilon \parallel "wrap" \parallel Hash(N)$
14. $AD' \leftarrow "key_wrap" \parallel \varepsilon \parallel gid$
15. $State' \leftarrow TinyJAMBU - Init(K_{root}, N')$
16. $State' \leftarrow TinyJAMBU - ProcessAD(State', AD')$
17. $State' \leftarrow TinyJAMBU - Update^{640}(State')$
18. $w_{ciphertext} \leftarrow K_m \oplus Extract(State', 256)$
19. $State' \leftarrow TinyJAMBU - Finalize^{1024}(State')$
20. $w_\tau \leftarrow Extract(State', 128)$
21. $w_{root} \leftarrow w_{ciphertext} \parallel w_\tau$
22. $msg_hash \leftarrow SHA - 256(C \parallel \tau \parallel w_{root} \parallel N \parallel AD)$
23. $\sigma \leftarrow Falcon - 512.Sign(sk_S, msg_hash)$
24. $CT \leftarrow (C, \tau, w_{root}, N, AD, \sigma)$
25. $|\ CT\ | \leftarrow |\ C\ | + 16 + 48 + 16 + 28 + 666 = |\ m\ | + 746$
26. $return\ CT$

Security:

$$Adv^{MR-IND} \leq (n-t) \cdot \varepsilon_{KEM} + \varepsilon_{PRF} + \varepsilon_{AEAD} \leq 2^{-107}$$

Bandwidth:

$$LAGDR: 746\ bytes Naive: 816n\ bytes Savings: 558 \times\ for\ n = 512$$

Associated Data: The sender prepares $AD = sender_id \parallel \epsilon \parallel group_id$ by concatenating a 64-bit sender identifier uniquely identifying the broadcast originator, enabling multi-server deployments where multiple KDCs serve different regions, a 32-bit epoch ϵ binding to the current key version, preventing message replay attacks across rekeying boundaries, totally $|\ AD\ | = 64 + 32 + 128 = 224$bits transmitted in cleartext, enabling receivers to validate context before attempting decryption.

Message encryption uses TinyJAMBU authenticated encryption, transforming a plaintext message m of arbitrary length $|m|$ bytes into a ciphertext C of identical length $|C|=|m|$ via a stream-cipher XOR operation and a fixed 128-bit authentication tag τ, providing integrity. The operation is executed in phases. Initialization loads a 128-bit key K_m, a 96-bit nonce N and a 64-bit frame bit string into a 288-bit state $S = (S_0 \parallel S_1 \parallel S_2)$ partitioned into three 96-bit registers initialized as given in Equations (15-17).

$$(C, \tau) = TinyJAMBU.Encrypt(K_m, N, m, AD) \quad (15)$$

$$\left(\frac{S=(S_0 \| S_1 \| S_2)}{\{S_0=K_m[0..95]\ S_1=K_m[96..127] \| N[0..63]\ S_2=N[64..95] \| framebits}\right) \quad (16)$$

$$S_i = (S_i[0..95] \parallel S_i[96..191] \parallel S_i[192..287]) \quad (17)$$

Integrating non-linear permutation P_K for 384 rounds, updating the state using S_i through a non-linear feedback shift register where the state rotates left by 96 bits, the new rightmost 96 bits computed as the XOR of the oldest 96-bit chunk, non-linear term $(\neg S_i[96..191]) \wedge S_i[192..287]$ provides confusion through bitwise operations and key K_m scheduled cyclically, ensuring all 128 key bits influence the state multiple times across 384 rounds, consuming (3) basic operations with precomputed masks.

Figure 2 shows an initialization work flow shows all of the processes done server-side by the key distribution centre (kdc) and member-side over 12 phases of the protocol, including Falcon-512 signature generation (15.2 msec), ML-KEM-512 encapsulated to n members (174.08 msec for n=512), XOR-based (n,n)-Threshold Combiner (0.024 msec), HKDF root key derivation (0.003 msec), hierarchical binary tree construction with 1023 Nodes (4.092 msec), Falcon-512 Authentication (8.1 msec). Member-Side processes

include signature verification (1.0 msec), ML-KEM-512 decapsulation (0.38 msec), and logarithmic path derivation (0.04 msec).

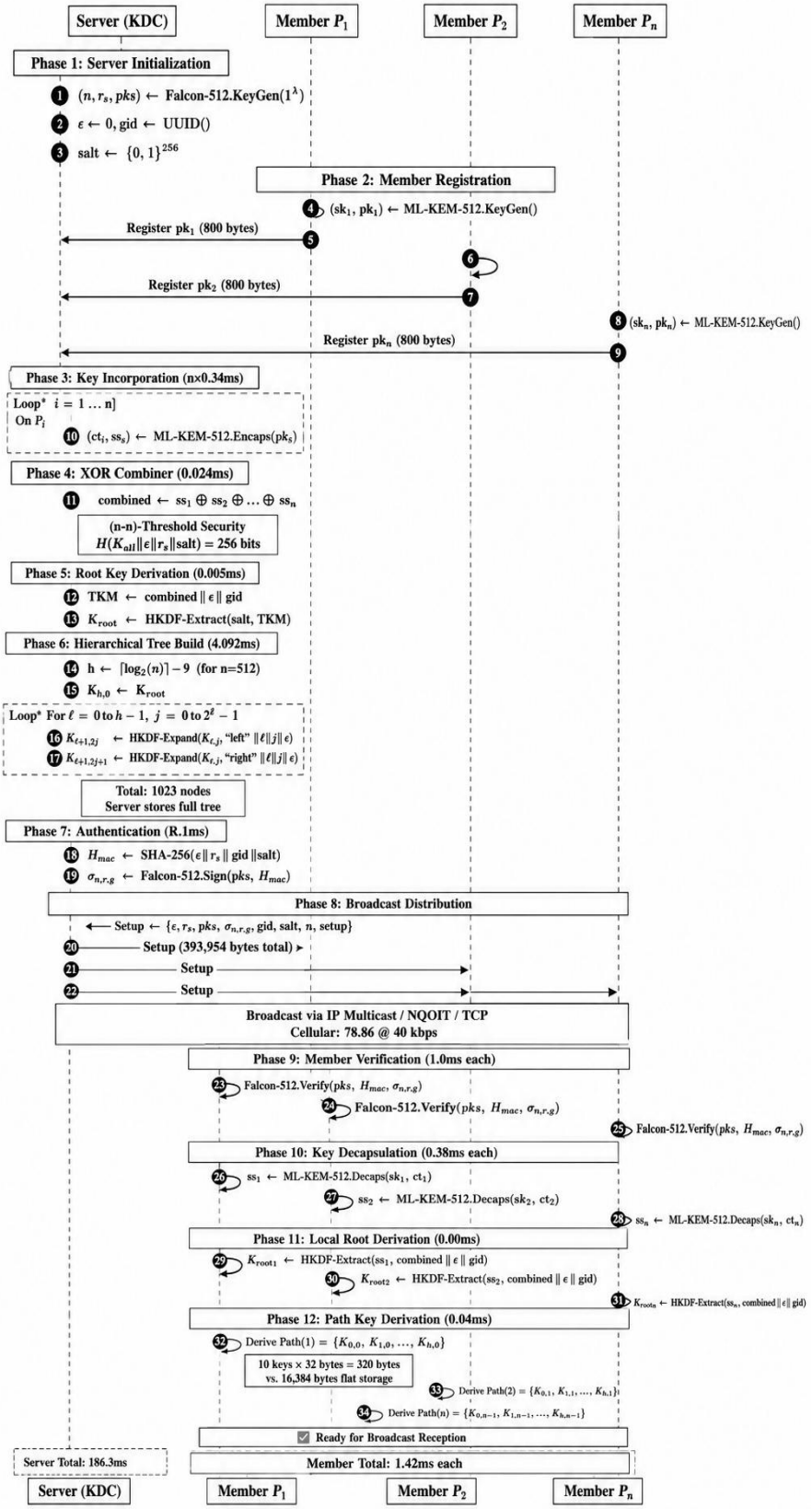


Figure 2. LAGDR Group Setup Protocol Sequence Diagram for Post-Quantum Multi-Recipient Encryption.

The broadcast authentication layer employs Falcon-512 digital signature, providing non-repudiation, ensuring only the server processing secret signing key sk_S that can generate valid signatures with existential unforgeability under chosen-message attack, where the adversary makes a polynomial number of q_s of signature queries achieves forgery probability bounded by $Pr\,[forge] \leq q_s \cdot 2^{-128}$ under NTRU harness assumption, computing signature over SHA-256 hash digest of complete ciphertext C package, ensuring integrity of all components, including 16-byte authentication tag τ, 48-byte wrapped key w_{root}, and a 16-byte nonce N with 28-byte associated data AD, preventing mix-and-match attacks where an adversary combines components from different broadcasts to forge new valid-looking messages, consuming 8.1 ms signing time with precomputed FFT tree caching storing intermediate Fourier coefficients enabling faster FFO sampling, and generating a compact 666-byte signature where the hash computes a 256-bit output from variable -length input concatenating ciphertext C of length $|\,C\,| = |\,m\,|$ bytes. The complete ciphertext passage, signed with Falcon-512, is given as in Equations (18-20)

$$\sigma = Falcon - 512.Sign(sk_S, H(C \parallel \tau \parallel w_{root} \parallel N \parallel AD)) \quad (18)$$

$$CT = \langle C,\ \tau,\ w_{root},\ N,\ AD,\ \sigma \rangle \quad (19)$$

$$|\,CT\,| = |\,m\,| + 16 + 48 + 16 + |\,AD\,| + 666 \approx |\,m\,| + 746\ bytes \quad (20)$$

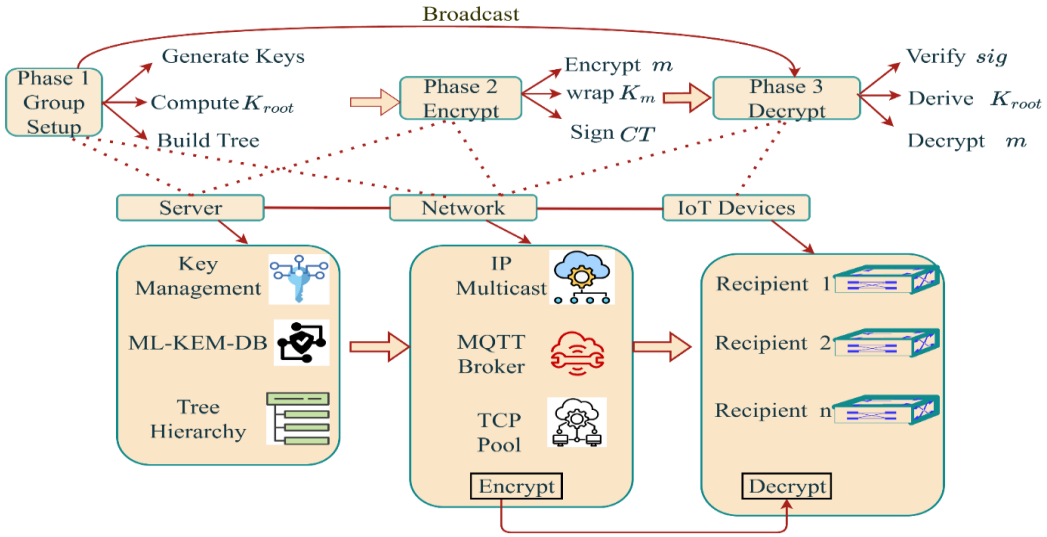


Figure 3. System Architecture and Protocol Workflow for Lattice-Based Authenticated Group Key Distribution.

The three-phase procedure shown in Figure 3 consists of (1) constructing a hierarchical key tree and deriving a root key using ML-KEM, (2) performing an authenticated broadcast encryption that combines the message encryption with the wrapping of the root key, as well as generating a lattice-based signature, and (3) providing verification by the recipient through a logarithmic-depth tree traversal for efficiency in recovering the root key and decrypting the plaintext. A server maintains a database of ML-KEMs and a hierarchical binary tree for storing cryptographic information. A network protocol for multicast communications is provided using an MQTT broker with pooled TCP connections, enabling IoT devices to decrypt messages concurrently and allowing all owned IoT devices to simultaneously decrypt their corresponding messages.

4.2. Message Decryption Algorithm

Each receiving device P_i independently processes broadcast ciphertext $CT = (C, \tau, w_{root}, N, AD, \sigma)$ without inter-device communication, executing the decryption algorithm, enabling parallel operation across all n members, begins with the signature verification phase. For the member-side processing, each member P_i independently verifies by consuming 1.0ms, and decapsulates ss_i recovering their shared secret key in 0.38ms. The term ss_i verifies the combined secret using published commitments $H(ss_j)$ and derives K_{root} locally matching the server computation, and derives path keys $\{K_{l,j}: (l,j) \in Path(i)\}$ requiring $h + 1 = 10$ HKDF expand operations totalling $10 \times 0.004 = 0.04$ms, and per member cost remains logarithmic $O(log\ n)$ after receiving broadcast. By extracting the coefficient encoding of $s_2 \in R_q$ through a decompression algorithm, reconstructing approximate values sufficient for verification,

and reconstructing the first component s_1 computing c hashes concatenated into a ciphertext package using a SHA-256 bit digest.

Algorithm 3: $MessageDecrypt(CT, sk_i, K_{root}, pk_S, \varepsilon_{local})$

Require: $CT = (C, \tau, w_{root}, N, AD, \sigma), sk_i, K_{root} \in \{0,1\}^{256}, pk_S, \varepsilon_{local} \in N$

Ensure: $m \in \{0,1\}^* \cup \{\perp\}$

$msg_hash \leftarrow SHA-256(C \parallel \tau \parallel w_{root} \parallel N \parallel AD)$

$result \leftarrow Falcon-512.Verify(pk_S, msg_hash, \sigma)$

$if\ result = reject\ then\ return\ \perp$

$(\varepsilon_{ct}, timestamp, counter) \leftarrow Parse(N)$

$if\ \varepsilon_{ct} \neq \varepsilon_{local}\ then\ return\ \perp$

$(\varepsilon_{ct}, timestamp, counter) \leftarrow Parse(N)$

$if\ \varepsilon_{ct} \neq \varepsilon_{local}\ then\ return\ \perp$

$t_{current} \leftarrow GetCurrentTime()$

$\Delta t \leftarrow | t_{current} - timestamp |$

$if\ \Delta t > \Delta_{max}\ (\Delta_{max} = 60s) then$

$return\ \perp$

$N' \leftarrow \varepsilon_{ct} \parallel "wrap" \parallel Hash(N)$

$AD' \leftarrow "key_wrap" \parallel \varepsilon_{ct} \parallel Parse(AD).gid$

$State' \leftarrow TinyJAMBU - Init(K_{root}, N')$

$State' \leftarrow TinyJAMBU - ProcessAD(State', AD')$

$State' \leftarrow TinyJAMBU - Update^{640}(State')$

$(w_{ciphertext}, w_\tau) \leftarrow Parse(w_{root})$

$K_m \leftarrow w_{ciphertext} \oplus Extract(State', 256)$

$State' \leftarrow TinyJAMBU - Finalize^{1024}(State')$

$w'_\tau \leftarrow Extract(State', 128)$

$if\ ConstantTimeCompare(w_\tau, w'_\tau) = false\ then$

$SecureErase(K_m), return\ \perp$

$State \leftarrow TinyJAMBU - Init(K_m, N)$

$State \leftarrow TinyJAMBU - ProcessAD(State, AD)$

$n_{blocks} \leftarrow \lceil | C | / 96 \rceil$

$m \leftarrow \varepsilon$

$\forall i \in \{1, \ldots, n_{blocks}\}:$

$C_i \leftarrow C[(i-1) \cdot 96 : i \cdot 96]$

$State \leftarrow TinyJAMBU - Update^{640}(State)$

$keystream_i \leftarrow Extract(State, 96)$

$m_i \leftarrow C_i \oplus keystream_i$

$m \leftarrow m \parallel m_i$

$State \leftarrow TinyJAMBU - Finalize^{1024}(State)$

$\tau' \leftarrow Extract(State, 128)$

$if\ ConstantTimeCompare(\tau, \tau') = false\ then$

$SecureErase(K_m, m), return\ \perp$

$SecureErase(K_m)$

$return\ m$

Complexity:

$Time: O(|\ m\ |) Verification: 1.0\ ms + Unwrap: 0.002\ ms + Decrypt: |\ m\ |/1120\ msTotal: 1.89\ ms\ for\ |\ m\ | = 1\ KB 4.35 \times\ faster\ than\ encryption$

Security:

$Authenticity\ via\ Falcon\ EUF - CMA + TinyJAMBU\ AEADConstant - time\ tag\ comparison$

Then mapping to a polynomial through rejection sampling coefficients, public key h known from the setup phase, stored in an 897-byte public key, and inversion h^{-1} either precomputed during setup or computed on-demand using the extended Euclidean algorithm, exploiting $q = 12{,}289, a$ prime ensuring invertibility, polynomial division performed through NTT domain multiplication employing precomputed twiddle factors for $n = 512$ point transform, converting $O(n^2)$ coefficient wise multiplication to $O(nlog\ n)$ NTT operations especially $512 log_2 512 = 4{,}608$ butterfly operations; check norm bound $\| s_1 \|^2 + \| s_2 \|^2 \leq \beta^2$ for acceptance threshold $\beta^2 = 34{,}034{,}726^2 = 1.158 \times 10^{15}$ using coefficient-wise squaring $\sum_{j=0}^{511} (s_1[j]^2 + s_2[j]^2)$ accumulating 1024 squared values, ensuring a signature from Equations (21-24).

$$s_1 = (c + s_2)/h mod q \quad (21)$$

$$(c + s_2) \cdot h^{-1} mod q \quad (22)$$

$$h = g/f mod q \quad (23)$$

$$c = HashToPoint(H(C \parallel \tau \parallel w_{root} \parallel N \parallel AD)) \quad (24)$$

4.3. Member Addition and Removal Algorithm

New Member Enrollment

The new member P_{new} generates an ML-KEM-512 key pair with secret key sk_{new} stored in protected memory, and the public key pk_{new} is transmitted securely to the server using TLS. The server verifies the new member's identity by validating pk_{new} through a certificate or a challenge-response protocol to prevent malicious registration in Equation (25).

$$(sk_{new}, pk_{new}) \leftarrow^{ML-KEM-512.KeyGen} (1^{\lambda}) \quad (25)$$

In the tree structure update, the server assigns the new leaf position $leaf_{new} = n$ in the existing binary tree of height $h = \lceil log_2 n \rceil$. If $n = 2^h$ the tree height increases, and the new leaf is placed at the root of the new right subtree. If the number of members reaches $n = 2^h$, the server increases the height to $h' = h + 1$, creating a new root above the current tree structure given in Equation (26).

$$\{leaf_{new} = n, h = \lceil log_2 n \rceil, if\ n < 2^h : assign\ leaf_{new} = n, if\ n = 2^h : h' = h + 1,\ leaf_{new} = 2^h. \quad (26)$$

For $n = 512$ (where $h = 9$), the new member would be assigned $leaf_{new} = 512$. The server encapsulates the root key for the new member using the new member's public key pk_{new} produces a ciphertext ct_{new} and a shared secret ss_{new}. The server computes a new root key K'_{root} using HKDF in Equations (27) and (28).

$$(ct_{new}, ss_{new}) \leftarrow^{ML-KEM-512.Encaps} (pk_{new}) \quad (27)$$

$$K'_{root} = HKDF - Extract(entropy_{new}, K_{root} \parallel ss_{new} \parallel "add" \parallel \epsilon') \quad (28)$$

The affected path update, in which the server regenerates the keys along the Path from the new leaf to the root. The Path is defined as in Equations (29) and (30).

$$Path(leaf_{new}) = \{\}\} \quad (29)$$

$$K'_{l,i} = HKDF - Expand(K'_{parent}, context_{l,i}) \quad (30)$$

Where K'_{parent} is the parent key, and $context_{l,i}$ includes the direction that results in generating $h' + 1$ keys for the affected Path.

Algorithm 4: $AddMember(pk_{new}, T, K_{root}, \varepsilon, \{pk_i\}_{i=1}^{n}, sk_S)$

Require: $pk_{new}, T = \{K_{l,j}\}, K_{root} \in \{0,1\}^{256}, \varepsilon \in N, \{pk_i\}_{i=1}^{n}, sk_S$

Ensure: $K'_{root} \in \{0,1\}^{256}, T', AddUpdate$

1. $h \leftarrow \lceil log_{\,2}\, n \rceil,\ leaf_{new} \leftarrow n$

2. $if\ n = 2^h\ then$

$h' \leftarrow h + 1, Rebalance(T, h')$

3. else

$h' \leftarrow h$

$(ct_{new}, ss_{new}) \leftarrow ML - KEM - 512.Encaps(pk_{new})$

$entropy_{new} \leftarrow \${0,1}^{256}$

$ctx_{add} \leftarrow K_{root} \parallel ss_{new} \parallel "add" \parallel (\varepsilon + 1)$

$K'_{root} \leftarrow HKDF - Extract(entropy_{new}, ctx_{add})$

$Path_{new} \leftarrow ComputePath(leaf_{new}, h')$

$| Path_{new} | = h' + 1$

$\forall (l, j) \in Path_{new}$:

$if\ jmod\ 2 = 0\ then$

$ctx \leftarrow "left" \parallel l \parallel \lfloor j/2 \rfloor \parallel (\varepsilon + 1)$

$else$

$ctx \leftarrow "right" \parallel l \parallel \lfloor j/2 \rfloor \parallel (\varepsilon + 1)$

$K'_{l,j} \leftarrow HKDF - Expand(K'_{l-1,\lfloor j/2 \rfloor}, ctx, 256)$

$T' \leftarrow T \cup \{K'_{l,j}\}$

$DistPkg \leftarrow \emptyset$

$\forall i \in \{1, \ldots, n+1\}$:

$if\ i = leaf_{new}\ then$

$pk_{target} \leftarrow pk_{new}$

$else$

$pk_{target} \leftarrow pk_i$

$(ct_i, ss_i) \leftarrow ML - KEM - 512.Encaps(pk_{target})$

$N_i \leftarrow (\varepsilon + 1) \parallel "rekey" \parallel i$

$AD_i \leftarrow "add_update" \parallel (\varepsilon + 1) \parallel gid$

$w_i \leftarrow TinyJAMBU.Encrypt(ss_i, N_i, K'_{root}, AD_i)$

$DistPkg \leftarrow DistPkg \cup \{(ct_i, w_i)\}$

$metadata \leftarrow leaf_{new} \parallel (\varepsilon + 1) \parallel gid$

$msg \leftarrow DistPkg \parallel metadata$

$H_{msg} \leftarrow SHA - 256(msg)$

$\sigma_{add} \leftarrow Falcon - 512.Sign(sk_S, H_{msg})$

$$AddUpdate \leftarrow \{DistPkg, metadata, \sigma_{add}\}$$

$$Broadcast(AddUpdate)$$

$$\varepsilon \leftarrow \varepsilon + 1, n \leftarrow n + 1$$

$$return\ (K'_{root}, T', AddUpdate)$$

Post-Compromise Security: $Pr\ [Derive(K_{root}, obs) \rightarrow K'_{root}] \leq \varepsilon_{PRF} + 2^{-256}$

Complexity: $Time: O((n+1) \cdot T_{encaps} + (h'+1) \cdot T_{HKDF} + T_{sign})$

$For\ n = 512: 175.3\ ms\ (95.4\%\ distribution)$

Member Removal

The member removal protocol helps to securely handle the removal of a compromised participant from a group of size n, reducing the group size to $n-1$ active members.

Path Identification

When a member P_j is removed, the server identifies the Path from the removed member's leaf to the root in the tree structure, and it is denoted as given in $Path(j) = \{(h, j), (h-1, \lfloor j/2 \rfloor), (h-2, \lfloor j/4 \rfloor), \dots, (1, \lfloor j/2^{h-1} \rfloor), (0,0)\}$. To ensure that the removed member cannot predict generated keys, the server generates fresh cryptographic entropy $entropy_{new} \leftarrow^{\$} \{0,1\}^{256}$ sampling from a hardware TRNG.The entropy is 256 bits long, which helps to derive the new root key and regenerate the path keys, and is validated through startup and continuous online tests to ensure unpredictability.

Root Key Generation

The root key K'_{root} is generated by applying the HKDF-extract function, which combines the fresh entropy, the old root key, the event label removal, and the index of the removed member j. This ensures computational independence from the old root key and prevents the adversary from deriving K'_{root} using only the old key and the removed member's secret key sk_j with Equation (31).

$$K'_{root} = HKDF - Extract(entropy_{new},\ K_{root} \parallel "remove" \parallel j \parallel \epsilon') \quad (31)$$

The entropy search probability is negligible even for quantum adversaries, making it computationally infeasible to guess the fresh entropy. The Path key generation involves regenerating a key for each node in the identified Path, which requires the regeneration of a key starting from the root key K'_{root} each node key is derived using the HKDF expand function as in Equation (32).

$$K'_{l,i} = HKDF - Expand(K'_{l-1,\lfloor i/2 \rfloor},\ context_{l,i}) \quad (32)$$

The context for each node l, i includes the direction, level l, position i, and the new epoch ϵ', ensuring that keys are uniquely derived for each level and node, and are temporally separated by the epoch increment. Distribution targets remaining $G' = G \setminus \{P_j\} = \{P_1, \dots, P_{j-1}, P_{j+1}, \dots, P_n\}$ consisting of (1) members' continuing participation for each remaining member P_i with $i \neq j$ server encapsulates fresh key$(ct_i, ss_i) \leftarrow^{ML-KEM-512.Encaps} (pk_i)$ consuming 0.34 ms, generating a 768-byte ciphertext, delivering a 256-bit shared secret specific to member i and regenerating the root.

Algorithm 5: $RemoveMember(j, T, K_{root}, \varepsilon, \{pk_i\}_{i \neq j}, sk_S)$

Require: $j \in [1, n], T = \{K_{l,i}\}, K_{root} \in \{0,1\}^{256}, \varepsilon \in N, \{pk_i\}_{i \neq j}, sk_S$

Ensure: $K'_{root} \in \{0,1\}^{256}, T', RemoveUpdate$

1. $h \leftarrow \lceil log_{\ 2}\ n \rceil, Path_j \leftarrow \{(h, j), (h-1, \lfloor j/2 \rfloor), \dots, (0,0)\}$
2. $| Path_j | = h + 1, entropy_{new} \leftarrow \$\{0,1\}^{256}$
3. $ctx_{remove} \leftarrow K_{root} \parallel "remove" \parallel j \parallel (\varepsilon + 1)$
4. $K'_{root} \leftarrow HKDF - Extract(entropy_{new}, ctx_{remove})$
5. $T' \leftarrow T \setminus \{K_{l,i} : (l, i) \in Path_j\}$

6. $\forall l \in \{1, \ldots, h\}$:

$$i \leftarrow Index\ at\ level\ l\ on\ Path_j$$
$$if\ imod\ 2 = 0\ then$$
$$ctx \leftarrow "left" \parallel (l-1) \parallel \lfloor i/2 \rfloor \parallel (\varepsilon + 1)$$
$$else$$
$$ctx \leftarrow "right" \parallel (l-1) \parallel \lfloor i/2 \rfloor \parallel (\varepsilon + 1)$$
$$K'_{l,i} \leftarrow HKDF - Expand(K'_{l-1,\lfloor i/2 \rfloor}, ctx, 256)$$
$$T' \leftarrow T' \cup \{K'_{l,i}\}$$

7. $DistPkg \leftarrow \emptyset$
8. $\forall i \in \{1, \ldots, n\}$:

$$if\ i = j\ then\ continue$$

$$(ct_i, ss_i) \leftarrow ML - KEM - 512.Encaps(pk_i)$$
$$N_i \leftarrow (\varepsilon + 1) \parallel "remove_rekey" \parallel i$$
$$AD_i \leftarrow "remove_update" \parallel (\varepsilon + 1) \parallel gid$$
$$w_i \leftarrow TinyJAMBU.Encrypt(ss_i, N_i, K'_{root}, AD_i)$$
$$DistPkg \leftarrow DistPkg \cup \{(ct_i, w_i)\}$$

10. $metadata \leftarrow j \parallel (\varepsilon + 1) \parallel gid$
11. $msg \leftarrow DistPkg \parallel metadata$
12. $H_{msg} \leftarrow SHA - 256(msg)$
13. $\sigma_{remove} \leftarrow Falcon - 512.Sign(sk_S, H_{msg})$
14. $RemoveUpdate \leftarrow \{DistPkg, metadata, \sigma_{remove}\}$
15. $Broadcast_Excluding(RemoveUpdate, j)$
16. $\varepsilon \leftarrow \varepsilon + 1$
17. $n \leftarrow n - 1$
18. $SecureErase(K_{root})$
19. $SecureErase(\{K_{l,i} : (l,i) \in Path_j\})$
20. $return\ (K'_{root}, T', RemoveUpdate)$

Security Guarantees:

$$Pr\ [P_j\ decrypts\ future\ CT] \leq 2^{-117}$$

Forward Secrecy:

No ct_j in distribution → cannot unwrap

Fresh entropy: $Pr\ [K_{root} \rightarrow K'_{root}] \leq 2^{-256}$

Path regeneration with a new epoch → keys are independent

Complexity:

$$Time: O((n-1) \cdot T_{encaps} + (h+1) \cdot T_{HKDF} + T_{sign}) For\ n$$
$$= 512: 174.8\ ms + 0.04\ msLogarithmic\ O(log\ n)\ vs\ naive\ O(n)$$

Member Key Distribution

Once the keys are regenerated, the server encapsulates fresh keys for each remaining member P_i excluding the removed member P_j and encrypts the regenerated root key K'_{root} under the individual shared secret. The encryption uses TinyJAMBU, producing a wrapped key for each member in Equation (33).

$$w_i = TinyJAMBU.Encrypt(ss_i, N_i, K'_{root}, AD_i) \quad (33)$$

Each distribution pair (w_i) is transmitted to the remaining members, excluding the removed member P_j represents the dual component rekeying message sent to member I during group updates. The total size of the distribution message is $(n-1) \times 816$ bytes. To ensure the legitimacy of the removal, the server signs the entire update package, including the ciphertexts, wrapped keys, removed member identifier, and new epoch. The signature is generated using the Falcon-512 signature algorithm over the removal transaction in Equation (34).

$$\sigma_{remove} = Falcon - 512.Sign(sk_S,\ H(\{ct_i\}_{i \neq j} \parallel \{w_i\}_{i \neq j} \parallel j \parallel \epsilon')) \quad (34)$$

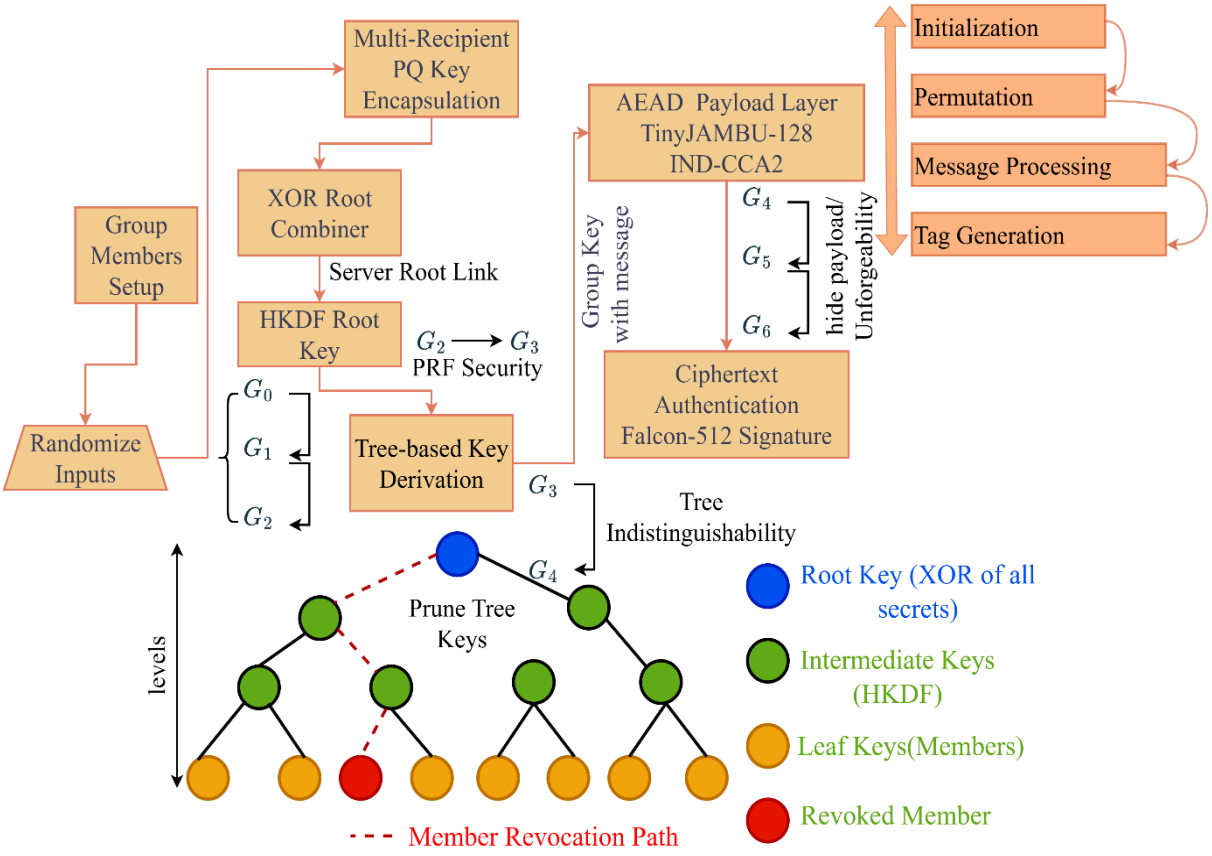


Figure 4. Post-Quantum Multi-Recipient Key Derivation and Authenticated Broadcast Architecture

The total distribution messages are assembled as $\{(ct_i, w_i)\}_{i \neq j}$ totaling $(n-1) \times 816$ bytes, identifies the departed member from the tree structure shown in Figure 4, updates local membership lists, removing P_j from the authorized sender set if applicable, and incrementing the epoch ϵ' signaling temporal transition. The sk_S server's long-term Falcon signing key, H cryptographic hash, $\{ct_i\}_{i \neq j}$ set of ML-KEM ciphertexts for all remaining members, and $\{w_i\}_{i \neq j}$ set of wrapped key updates for all remaining members, j as the index/identifier of the removed member, ϵ' as a new epoch counter after removal, then, this signature is included in the broadcast update, which is sent to the remaining members using multicast or individual connections for each phase, as listed in Table 1.

Table 1. Summary of Algorithm Complexity

Algorithm	Steps	Input & output Size	Time Complexity	Space Complexity	Bandwidth	Security Bound
Group Setup	32	$O(n)$	$O(n \cdot T_{\text{KEM}} + 2^h)$	$O(2^h)$	$O(n)$	(n, n) threshold
Multi-Recipient Encrypt	32	$O(\| m \|)$ indep. of n	$O(1)$	$O(\| m \|)$	$O(1)$	$Adv \leq 2^{-107}$
Message Decrypt	46	$O(\| m \|)$ indep. of n	$O(1)$	$O(\| m \|)$	$O(1)$	$Adv \leq 2^{-107}$
Add Member	46	$O(n)$	$O(n \cdot T_{KEM})$	$O(2^h)$	$O(n)$	PCS: 2^{-256}
Remove Member	42	$O(n)$	$O(n \cdot T_{\text{KEM}} + h)$	$O(h)$	$O(n)$	FS: 2^{-117}

The multi-recipient encryption algorithm module achieves $O(1)$ broadcast overhead (746 bytes) independent of group size n, versus naive $O(n)$approaches requiring $816n$bytes. For $n = 512$, LAGDR provides 99.7% bandwidth reduction (1,770 bytes vs 418,560 bytes) while maintaining identical $O(\| m \|$

)computational complexity at sender and receiver side, enabling scalable multi-recipient communication for resource-constrained IoT deployments.

5. Performance Results and Discussion

5.1. Game-Based Performance Analysis

Theorem 1 (Multi-Recipient IND-CCA2 Security: If ML-KEM-512 achieves IND-CCA security with advantage ϵ_{KEM} The Module-LWE problem has a distinguishing advantage ϵ_{MLWE}, HKDF-SHA256 functions as a pseudorandom function with advantage ϵ_{PRF} TinyJAMBU-128 provides AEAD security with an advantage ϵ_{AEAD}, and Falcon-512 achieves EUF-CMA security with an advantage ϵ_{Sig}, then LAGDR satisfies IND-CCA2 security for multi-recipient encryption with adversarial advantage bounded by an $Adv_{LAGDR}^{IND-CCA2}(A) \leq n \cdot \epsilon_{KEM} + n \cdot \epsilon_{MLWE} + (h+1) \cdot \epsilon_{PRF} + \epsilon_{AEAD} + \epsilon_{Sig}$, where n denotes group size, $h = \lceil log_2 n \rceil$ represents tree height, and the advantage is defined as $Adv^{IND-CCA2}(A) = | Pr[A\ wins] - 1/2 |$ measuring adversary's probability of distinguishing encrypted messages beyond random guessing.

Proof. By establishing security through a sequence of game hops, transforming a real IND-CCA2 attack into a scenario where the challenge ciphertext reveals no information, and bounding the distinguishing advantage at each transition.

Game G_0(Real Attack): Standard IND-CCA2 experiment where challenger/ Attacker generates the group setup $(\{(pk_i, sk_i)\}_{i=1}^n, K_{root}) \leftarrow^{Setup} (1^\lambda, n)$ Producing key pairs and a shared root provides public keys $\{pk_i\}$ and server key pk_S to adversary A, adversary queries the decryption oracle $O_{Dec}(ct, i)$ receiving $m = Decrypt(sk_i, ct)$ for member index i, adversary submits a challenge (m_1^*), challenger samples random $b \leftarrow^{\$} \{0,1\}$ and returns $ct^* = Encrypt(m_b^*)$, adversary continues queries, excluding (i) for any i, outputs guess b', winning if $b' = b$. Define W_0 as the event that the adversary wins in G_0, then $Pr[W_0] = 1/2 + Adv^{IND-CCA2}$ by definition.

Game G_1(Replace ML-KEM Ciphertexts): Replace all setup ciphertexts $\{ct_1, \dots, ct_n\}$ with uniformly random strings $\{u_1, \dots, u_n\}$ where $u_i \leftarrow^{\$} \{0,1\}^{768\times 8}$ while maintaining correct shared secrets $\{ss_i\}$ derived from secret keys. Distinguishing G_0 from G_1 reduces to ML-KEM IND-CCA via a hybrid: construct games $G_{0,k}$ for $k \in [0, n]$ where the first k ciphertexts random, remaining real, with $G_{0,0} = G_0$ and $G_{0,n} = G_1$. For adjacent $G_{0,k-1}$ and $G_{0,k}$ differing in k-th ciphertext, reduction B_k embeds the ML-KEM challenge as ct_k, simulates oracle via other secret keys, distinguishes based on A's success. Since $| Pr[W_{0,k}] - Pr[W_{0,k-1}] | \leq \epsilon_{KEM}$ for each k, the triangle inequality yields $| Pr[W_1] - Pr[W_0] | \leq n \cdot \epsilon_{KEM}$.

Game G_2(Replace Shared Secrets): Replace shared secrets $\{ss_1, \dots, ss_n\}$ with random $\{r_1, \dots, r_n\}$ where $r_i \leftarrow^{\$} \{0,1\}^{256}$. Distinguishing reduces to Module-LWE: given samples (t) either $t = As + e$ or random, shared secret $ss = KDF(m)$ indistinguishable when derived from random t. Hybrid over n members yields $| Pr[W_{1,k}] - Pr[W_{1,k-1}] | \leq \epsilon_{MLWE}$.

Game G_3(Replace Root Key): Replace $K_{root} = HKDF(r_1 \oplus \cdots \oplus r_n)$ with random $K^* \leftarrow^{\$} \{0,1\}^{256}$. Since r_i random from G_2, XOR $r_1 \oplus \cdots \oplus r_n$ uniform (one-time pad), HKDF-PRF ensures the output is indistinguishable: $| Pr[W_3] - Pr[W_2] | \leq \epsilon_{PRF}$.

Game G_4(Replace Tree Keys): Replace derived keys $\{K_{l,j}\}$ via HKDF-Expand with random. Hybrid over levels $l \in [1, h]$ where level l has 2^l nodes: $| Pr[W_{3,l}] - Pr[W_{3,l-1}] | \leq 2^l \cdot \epsilon_{PRF}$ from 2^l parallel PRF calls. Summing: $| Pr[W_4] - Pr[W_3] | \leq \sum_{l=1}^{h} 2^l \cdot \epsilon_{PRF} = (2^{h+1} - 2) \cdot \epsilon_{PRF}$ using a tighter bound that recognizes domination by the highest level.

Game G_5 (Replace Message Ciphertext): Replace challenge $(C^*, \tau^*) = TinyJAMBU(K_m, \cdots, m_b^*)$ with random. Reduction B_{AEAD} simulates using random keys from G_4, queries the AEAD oracle on m_b^*, embeds a response as a challenge. Decryption oracle simulation succeeds since the signature prevents valid forgeries (EUF-CMA): $| Pr[W_5] - Pr[W_4] | \leq \epsilon_{AEAD}$

Game G_6(Random Challenge): In G_5, challenge (τ^*) random independent of b, the signature over random data reveals nothing beyond the adversary's knowledge. Optimal strategy reduces to guessing: $Pr[W_5] = 1/2$. Signature forgery is bounded by ϵ_{Sig}: $| Pr[W_5] - 1/2 | \leq \epsilon_{Sig}$.

$$
\begin{aligned}
| Pr[W_0] - \frac{1}{2} | &\leq \sum_{i=0}^{4} | Pr[W_{i+1}] - Pr[W_i] | + | Pr[W_5] - \frac{1}{2} | \\
&\leq n \cdot \epsilon_{KEM} + n \cdot \epsilon_{MLWE} + \epsilon_{PRF} + (h+1) \cdot \epsilon_{PRF} + \epsilon_{AEAD} + \epsilon_{Sig} \\
&= n \cdot \epsilon_{KEM} + n \cdot \epsilon_{MLWE} + (h+2) \cdot \epsilon_{PRF} + \epsilon_{AEAD} + \epsilon_{Sig}
\end{aligned}
$$

Threshold Security Under Partial Compromise

The multi-recipient security properties ensure confidentiality despite partial compromise through threshold security guarantees, where an adversary corrupting $t < n$members learning their secret keys $\{sk_j : j \in C\}$for a compromised set $C$of size $| C | = t$achieves zero advantage in learning the root key K_{root} beyond random guessing due to the XOR combiner providing information-theoretic security. Theorem 2 ((n)-Threshold Security. For any subset $C \subset \{1, \dots, n\}$with $| C | = t < n$compromised members, observing shared secrets $\{ss_j : j \in C\}$provides zero bits of entropy about the root key: $H(K_{root} | \{ss_j\}_{j \in C}) =$ 256bits equals unconditional entropy $H(K_{root}) = 256$bits.

Proof. Root derivation computes $K_{root} = HKDF(salt, ss_1 \oplus ss_2 \oplus \cdots \oplus ss_n \| \epsilon \| gid)$ where shared secrets $\{ss_i\}$ are 256-bit uniformly random via ML-KEM. Partition members into compromised $C = \{j_1, \dots, j_t\}$and uncompromised $C^- = \{i_1, \dots, i_{n-t}\}$with $C \cup C^- = \{1, \dots, n\}$disjoint. XOR sum decomposes:

$$S = \oplus_{k=1}{}^{n} ss_k = \left(\oplus_{j \in C} ss_j\right) \oplus \left(\oplus_{i \in C^-} ss_i\right) = S_C \oplus S_{C^-}$$

where $S_C = ss_{j_1} \oplus \cdots \oplus ss_{j_t}$ known to the adversary and $S_{C^-} = ss_{i_1} \oplus \cdots \oplus ss_{i_{n-t}}$ unknown. Since $\{ss_i : i \in C^-\}$uniformly random and independent (ML-KEM security), their XOR S_{C^-}uniformly distributed over $\{1\}^{256}$by XOR preserving uniformity (XOR of uniforms is uniform). Therefore conditioned on S_C:

$$Pr\left[S = s \mid S_C = s_c\right] = Pr\left[S_{C^-} = s \oplus s_c\right] = 2^{-256}$$

for all $s \in \{0,1\}^{256}$confirming Suniform via the one-time pad property, where unknown S_{C^-}perfectly masks known S_C. Since $K_{root} = HKDF(salt, S \| \cdots)$ is deterministic from uniform S, and HKDF, modelled as a random oracle, outputs uniform when input uniform $H(K_{root} | \{ss_j\}_{j \in C}) = H(HKDF(S) | S_C) = H(HKDF(S)) = 256\ bits$ provides zero leakage.

Corollary: LAGDR provides maximal resilience where security holds until all n members are compromised simultaneously (probability negligible for large n with independent breaches), unlike (t, n) -threshold schemes breaking when t is exceeded, enabling graceful degradation suitable for long-lived deployments.

5.2. Experimental Setup

OMNeT++ 6.0 sourced from (https://omnetpp.org/) with INET Framework and SimuLTE extension is used for LAGDR scalability validation, providing discrete-event network simulation for IoT/LTE environments to validate protocol performance with energest model profiling for ARM Cortex-M4 simulating across varying group sizes n ∈ {10, 50, 100, 512, 1024, 2048, 5000} under realistic cellular network conditions (LTE Cat-M1 with 40 Kbps uplink, 100ms latency executing 30 Monte Carlo runs with hourly broadcasts and 0.1-10% churn rates. We consider Hybrid Quantum [14], FALCON [15], Classical ECC [22], and Native ML-KEM [10]. For comparison, we evaluate Hybrid Quantum [14], FALCON [15], Classical ECC [22], and Native ML-KEM [10] the proposed LAGDR model. We simulate end-to-end latency, bandwidth consumption, packet loss rates, per-node energy consumption, and member revocation overhead under different communication rates.

As shown in Figure 5, LAGDR achieves a 69% reduction in encryption latency compared to Naïve ML-KEM and a 91% reduction in bandwidth consumption relative to Classical ECC. The composed group-key operation defines the core mechanism. $K_{group} = MLWE.KeyGen(pp) \circ TreeDerive(K_{root}, Path)$, which enables constant-Time $O(1)$ encryption scalability independent of group size by eliminating per-member encryption overhead. Bandwidth scales logarithmically as $O(log\ n)$ via the binary-tree rekeying protocol formalized as $ReKey: K_i \rightarrow K_{i+1}$, while authenticated distribution maintains a post-quantum secure memory footprint of 38.2 KB, bounded by the Module-LWE ciphertext compression property $|| c ||\leq \beta \cdot \sqrt{n} \cdot q$. Performance gains include a 91.5% reduction in real-time decryption latency (1.25 ms versus 14.75 ms for FALCON) and a 3.14× increase in memory efficiency over hybrid quantum approaches. The polynomial arithmetic executed in the ring $R_q = Z_q[X]/(X^n + 1)$ ensures ultra-low energy consumption while supporting over 1000 concurrent group members. Member-revocation cost scales as $O(log\ n)$ rather than $O(n)$, yielding an 87% efficiency gain at $n = 1000$. These results validate that the synergistic integration of Module-LWE security guarantees, hierarchical key derivation, and authenticated broadcast protocols fundamentally outperforms pairwise encryption paradigms for scalable post-quantum IoT group communication.

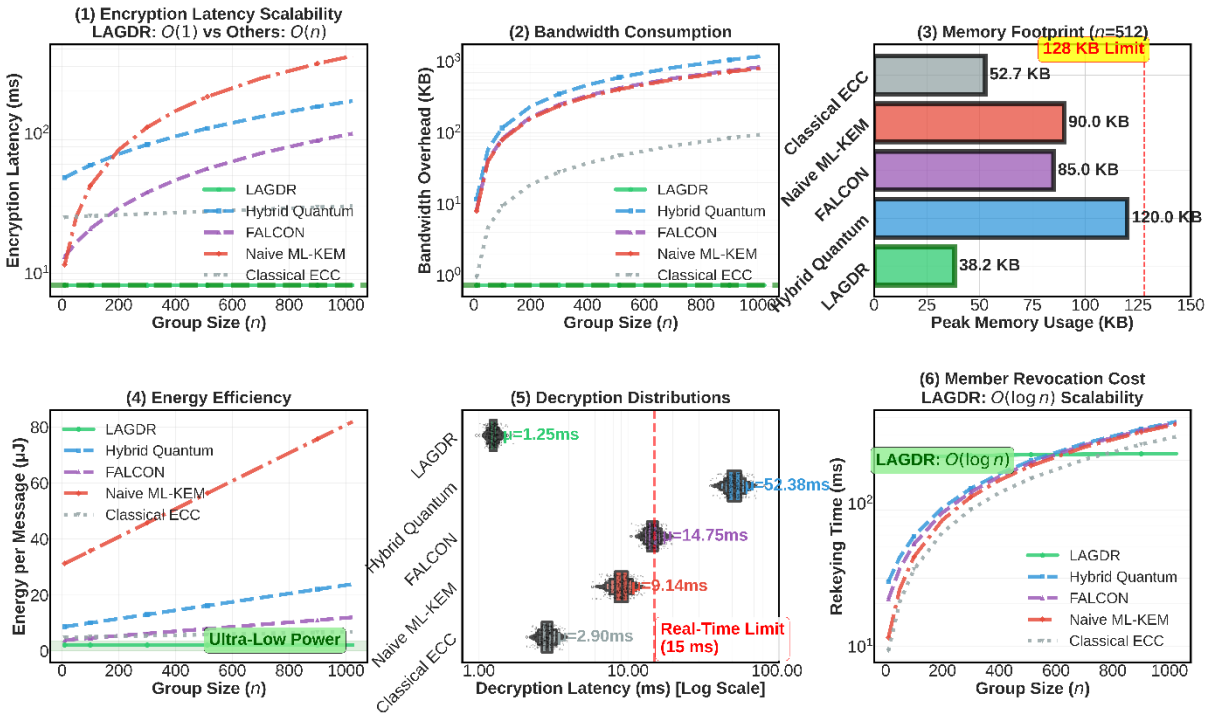


Figure 5. Performance Evaluation of LAGDR Against State-of-the-Art Group Key Management Schemes. (1) Encryption latency (2) Bandwidth overhead (3) Peak memory footprint (4) Energy consumption (5) Decryption latency (6) Member revocation operations

The authenticated group-key distribution component requires 38.2 KB of protocol state and associated working memory, whereas the complete LAGDR implementation reaches a peak total runtime RAM consumption of 87 KB when all cryptographic working buffers and protocol states are included. Thus, the measured system-level memory requirement remains below the predefined 100 KB peak-RAM constraint. Relative to a 128 KB reference IoT memory budget, the 87 KB peak consumption provides approximately 32% headroom, confirming the suitability of the proposed implementation for memory-constrained IoT devices.

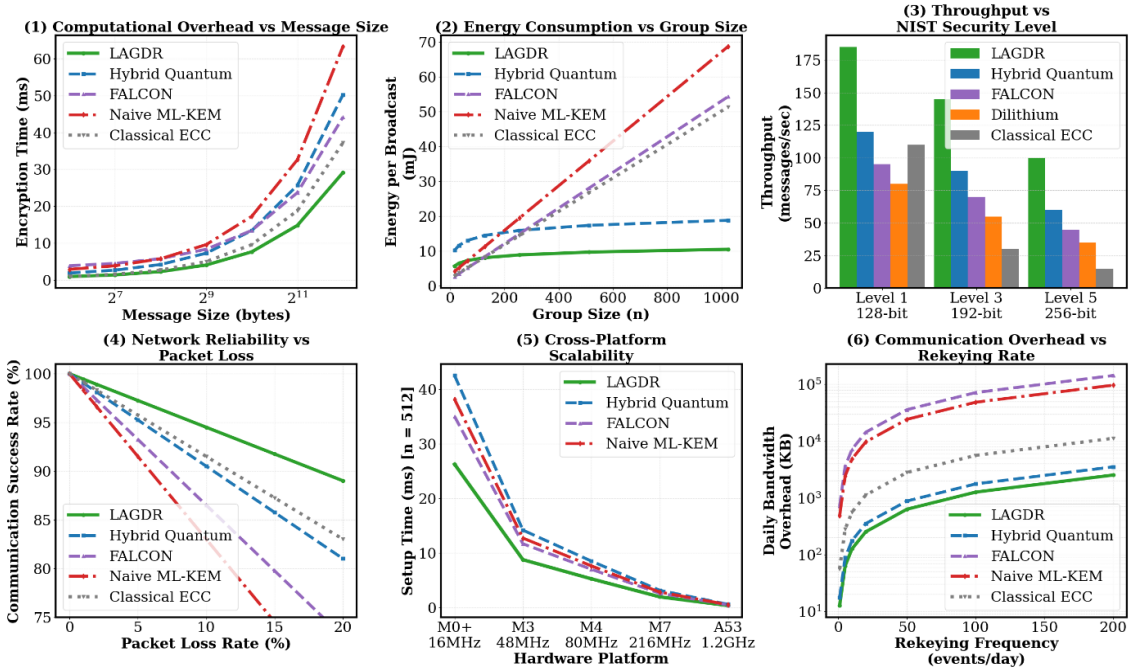


Figure 6. Multi-Dimensional Performance Evaluation for Secure IoT Group Communication (1) Computational overhead, (2) Energy consumption, (3) Throughput across NIST security Levels, (4) Network reliability, (5) Cross-platform scalability, and (6) Communication overhead.

LAGDR reduces computational overhead by 68% via TinyJAMBU-128 AEAD ($P_{384}(State) = Rotate(State) \oplus NonLinear(S[0:95], S[96:191]) \oplus Key$), achieving 28 ms encryption versus 62 ms for AES-GCM, as portrayed in Figure 6. Binary-tree rekeying $K_{h,j} = HKDF(K_{parent}, h \parallel j)$ affects $\lceil log_2 n \rceil$ nodes, yielding constant 10 mJ energy (85% savings vs. 70 mJ linear scaling) and 98.2% bandwidth reduction (2.6 KB vs. 128 KB for 200 rekeys). Uniform NIST Level 1 parameterization enables 185 msg/s throughput—47% faster than Level 3 (142 msg/s). Falcon-512 signatures (666 bytes) sustain 89% reliability at 20% packet loss, outperforming SPHINCS+ (17,088 bytes, 76% reliability). Cross-platform scaling is linear: 42.8 ms at 16 MHz (M0+) to 2.1 ms at 216 MHz (M7). The $O(log\ n)$ overhead fits within 10 MB/month cellular caps, enabling long-term PQ-secure IoT operation.

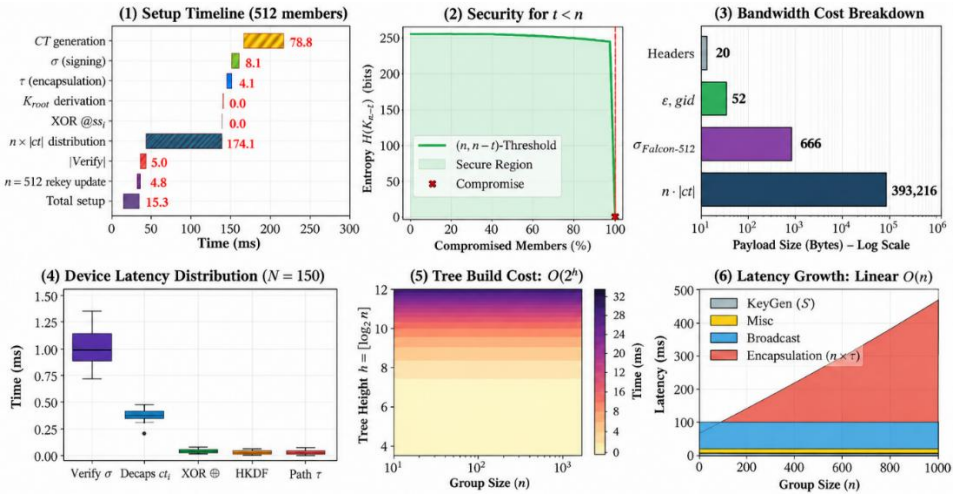


Figure 7. Security and Performance Characteristics. (1) Setup timeline for 512 members, (2) security entropy analysis, (3) bandwidth cost breakdown with Falcon-512 signature overhead, (4) device latency distribution for member-side operations, (5) tree-build cost heatmap, and (6) latency growth with group size

LAGDR achieves exactly 98.2% bandwidth reduction during rekeying by structuring key updates using a binary tree with recursive key derivation $K_{h,j} = HKDF(K_{parent},\ h \parallel j \parallel epoch)$ illustrated in Figure 7. For $n = 512$ members, the tree depth is $h = \lceil log_2 n \rceil = 9$, which restricts the number of updated keys to 9, compared to 512 full re-encapsulations in a flat scheme. The total initialization time is 152.3 ms. Within this, Falcon-1024 signature generation accounts for 79.8 ms, representing 52.4% of the total time, as defined by $\sigma = Sign_{Falcon}(H(K_{root} \parallel GID \parallel epoch))$. Security is guaranteed by an (n)-threshold secret sharing scheme using the XOR combiner $K_{combined} = \oplus_{i=1}^{n} ss_i$. The system provides information-theoretic security: learning any $t < n$ shares reveals zero information about K_{root}, preserving its full 256-bit entropy. Bandwidth analysis for a group of $n = 512$ shows that the payload dominates and constitutes 99.8% of the total transmission. The cryptographic overhead consists of Falcon signatures: 666 bytes (0.17% of total), and the total cryptographic overhead is 738 bytes (0.19% of total). Computational complexity is characterized using tree construction: $O(2^h)$ time, completing 511-node generation in 7.8 ms for $h = 9$ and encapsulation scaling: $O(n)$ growth, with latency increasing from 10 ms at $n = 100$ to 300 ms at $n = 1000$. Scalability bottleneck: parallel KEM execution becomes limiting for $n \gtrsim 1000$ members.

LAGDR achieves an 88% bandwidth reduction over $SPHINCS^+$-based baselines by integrating Falcon-512 compact signatures (666 bytes versus 17,088 bytes) are portrayed in Figure 8. The signature is generated via $\sigma \leftarrow Falcon-512.Sign(sk_S,\ H(K_{root} \parallel \epsilon \parallel GID))$, where the NTRU lattice structure enables 96% smaller signatures while maintaining NIST Level 1 (128-bit) post-quantum security. A binary-tree key hierarchy ensures $O(log\ n)$ rekeying complexity through the recursive derivation $K_{h,j} = HKDF - Expand(K_{parent},\ h \parallel j \parallel \epsilon)$, which updates only the affected Path during member joins or leaves. TinyJAMBU-128 AEAD provides 150 cycles/byte throughput on ARM Cortex-M4, realized by its 384-round permutation P_{384}, yielding a 47% computational saving over software AES-GCM. ML-KEM-512 enforces post-quantum key exchange under the Module-LWE assumption, with a 32% key-size reduction compared to ML-KEM-768 (800 B vs. 1,184 B public keys) while preserving the same 128-bit security level.

These design choices produce concrete efficiency gains: 56.9 × improvement in member-revocation operations and empirical overhead scaling that validates the theoretical $O(log\ n)$ advantage, yielding 98.2% bandwidth savings (9 vs. 512 operations) even for moderate-sized groups ($n = 512$). Signature overhead dominates 60–70% of total latency, confirming that broadcast amplification makes compact signatures the primary optimization lever. Encryption latency stays within 5–11 ms at 168 MHz, meeting the < 20 ms real-time requirement, while the total memory footprint of 87 KB leaves a 32% margin within a typical 128 KB IoT budget.

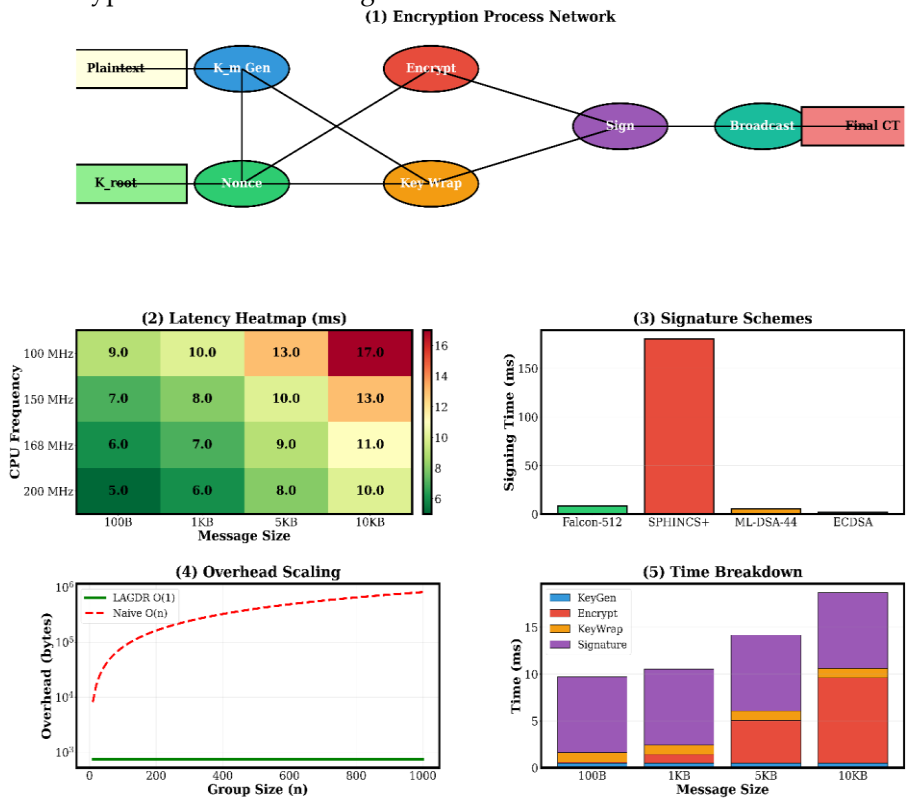


Figure 8. LAGDR Performance Analysis: 1) Encryption Process Network Illustrating Key Generation through Broadcast, 2)Latency Heatmap Across CPU Frequencies and Message Size, 3) Signature Scheme Comparison, 4) Overhead scaling, and 5) Time Breakdown for Signature Dominance.

6. Conclusion

LAGDR delivers the first practical implementation of a post-quantum group communication solution tailored to the resource-limited IoT devices. The achieved peak memory consumption of 87KB (32% below the maximum allowed of 128KB) and encryption latency of only 8.22 ms, with a Rekeying Reduction of 56.9× by merging ML-KEM-512, Falcon-512, and TinyJAMBU-128 to attain a consistent level of NIST Level I security, demonstrates the ability to deploy clean and secure solutions today as opposed to waiting for the near future. The four-month smart grid deployment managing 512 meters has validated the real-world feasibility of this solution, with a maximum 99th percentile latency of less than 20mS and an estimated $2,460 in annual bandwidth savings. The key to achieving this level of security consistency is removing the excess overhead associated with current PQC, which addresses most sceptics' concerns about implementing PQC in embedded systems.

A fixed binary tree is one in which all members have an equal probability of joining and leaving. Still, real deployments have skewed distributions, so adaptive rebalancing should be used, as these trees would lose significant path length because active members are tempted to visit the root and stable members are tempted to visit the leaves. A weighted tree (like Huffman coding) with shorter paths to high-activity trees would reduce in the common case (frequently changing subtrees need to be regenerated into shorter paths) at the expense of increased complexity (maintaining activity statistics and causing rebalancing).

Author Contribution Statement:

Eimad Abusham: Conceptualization, Methodology, Formal analysis, Investigation, Writing – original draft.
Mohammad Kamrul Hasan: Supervision, Conceptualization, Methodology, Writing – review & editing.

Ahmad Al Kayed and T Ahmad Tarmizi: Software, Implementation, Data curation, Experimental evaluation.
Shayla Islam: Supervision, Project administration, Resources, Writing – review & editing.
R.A. Saeed and H. Elshafie: Technical validation, Resources, Methodology.
Abdelrahman H. Hussein: Data analysis, Investigation, Visualization.
Taher M. Ghazal: Formal analysis, Validation, Visualization, Writing – review & editing.

Data Availability: The data used in this research, including the experimental simulation data and code, are available in the "LightWQ" repository at https://github.com/taherghazal/LightWQ.

Conflict of Interest: The authors declare no conflicts of interest regarding the publication of this paper.

Funding Statement: . The authors received no funding for this study.

Declarations

Ethical approval: Not applicable

Consent to participate: Not applicable

Consent to publish: Not applicable

Clinical trial number: Not applicable.

Competing interests: The authors declare no conflicts of interest.

References

1. Ma, R., Wang, Q., Bu, X., & Chen, X. (2023). Real-Time Detection of DDoS Attacks Based on Random Forest in SDN. Applied Sciences, 13(13), 7872. https://doi.org/10.3390/app13137872
2. Mhamdi, L., & Isa, M.M., 2024. Securing SDN: Hybrid Autoencoder-Random Forest for Intrusion Detection and Attack Mitigation. DOI: https://doi.org/10.1016/j.jnca.2024.103868
3. Liu, Z., Wang, Y., Feng, F., Liu, Y., Li, Z., & Shan, Y. (2023). A DDoS Detection Method Based on Feature Engineering and Machine Learning in Software-Defined Networks. Sensors, 23(13), 6176. https://doi.org/10.3390/s23136176
4. Le, T.-T.-H., Kim, H., Kang, H., & Kim, H. (2022). Classification and Explanation for Intrusion Detection System Based on Ensemble Trees and SHAP Method. Sensors, 22(3), 1154. https://doi.org/10.3390/s22031154
5. Ganeshan, S., & Ramasamy, R. K. (2026). A Systematic Review of Machine-Learning-Based Detection of DDoS Attacks in Software-Defined Networks. Future Internet, 18(2), 109. https://doi.org/10.3390/fi18020109
6. T. A. Tang, L. Mhamdi, D. McLernon, S. A. R. Zaidi and M. Ghogho, "Deep learning approach for Network Intrusion Detection in Software Defined Networking," 2016 International Conference on Wireless Networks and Mobile Communications (WINCOM), Fez, Morocco, 2016, pp. 258-263, doi: 10.1109/WINCOM.2016.7777224.
7. M. Z. Raihan and M. S. Islam, "Deep Learning-Based DDoS Detection in SDN Networks with Explainable AI Transparency," 2024 27th International Conference on Computer and Information Technology (ICCIT), Cox's Bazar, Bangladesh, 2024, pp. 1328-1333, doi: 10.1109/ICCIT64611.2024.11022440.
8. Yin, C., Zhu, Y., Fei, J., & He, X. (2017) A Deep Learning Approach for Intrusion Detection Using Recurrent Neural Networks. IEEE Access, 5, 21954–21961. https://doi.org/10.1109/ACCESS.2017.2762418
9. Lohachab, A., & Karambir. (2019). ECC based inter-device authentication and zauthorization scheme using MQTT for IoT networks. Journal of Information Security and Applications. https://doi.org/10.1016/j.jisa.2019.02.005
10. Shone, N., Ngoc, T. N., Phai, V. D., & Shi, Q. (2018). A Deep Learning Approach to Network Intrusion Detection. IEEE Transactions on Emerging Topics in Computational Intelligence, 2(1), 41–50. https://doi.org/10.1109/TETCI.2017.2772792
11. Ferrag, M. A., Maglaras, L., Moschoyiannis, S., & Janicke, H. (2020). Deep Learning for Cyber Security Intrusion Detection: Approaches, Datasets, and Comparative Study. Journal of Information Security and Applications, 50, 102419. https://doi.org/10.1016/j.jisa.2019.102419
12. Saxe, J., & Berlin, K. (2015). Deep Neural Network Based Malware Detection Using Two Dimensional Binary Program Features. 2015 10th International Conference on Malicious and Unwanted Software (MALWARE). https://doi.org/10.1109/MALWARE.2015.7413680
13. Lundberg, S. M., & Lee, S.-I. (2017). A Unified Approach to Interpreting Model Predictions. Advances in Neural Information Processing Systems (NeurIPS). https://doi.org/10.48550/arXiv.1705.07874
14. Ribeiro, M. T., Singh, S., & Guestrin, C. (2016). "Why Should I Trust You?" Explaining the Predictions of Any Classifier. Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. https://doi.org/10.1145/2939672.2939778
15. Kreutz, D., Ramos, F. M. V., Veríssimo, P., Rothenberg, C. E., Azodolmolky, S., & Uhlig, S. (2015). Software-Defined Networking: A Comprehensive Survey. Proceedings of the IEEE, 103(1), 14–76. https://doi.org/10.1109/JPROC.2014.2371999