TreeCast: Multi-Party Key Establishment Protocol for IoT Devices

Supriyo Banerjee[1] and Sayon Duttagupta[2]

[1] Indian Institute of Technology Madras, Chennai, India
supriyobanerjee537@gmail.com
[2] COSIC, KU Leuven, Leuven, Belgium
Sayon.Duttagupta@esat.kuleuven.be

Abstract. Secure communication in the Internet of Things (IoT) requires lightweight protocols that scale across unicast, multicast, and broadcast settings. Existing solutions typically depend on centralized gateways, which introduce single points of failure and scalability limitations. We propose *TreeCast*, a distributed group key establishment protocol that organizes devices into a binary tree of hashed Diffie–Hellman secrets, which naturally unifies unicast, multicast, and broadcast in a single scalable hierarchical structure. This design yields logarithmic rekey costs, efficient subtree derivation with efficient device addition and revocation in large and dynamic IoT environment and strong security properties including authentication, partial forward secrecy, and post-compromise recovery. We present a detailed security analysis and to demonstrate practical viability, implement TreeCast on an Arm Cortex-M33 using only software cryptography, without any hardware accelerators. Our measurements show that TreeCast performs key establishment and rekeying within realistic IoT timing and energy budgets, achieving sub-millisecond tree updates and low memory footprint. These results establish TreeCast as a lightweight, scalable, and deployable solution for secure communication in next-generation IoT networks.

1 Introduction

Large-scale Internet of Things (IoT) deployments increasingly operate without continuous human supervision, coordinating sensing and actuation across consumer, industrial, and urban environments. In such settings, devices must communicate with individual peers, selected groups, or entire networks. These interactions naturally occur in three modes: *unicast* (one-to-one), *multicast* (one-to-subset), and *broadcast* (one-to-all).

These communication patterns arise frequently in real systems. For example, a smart water meter may send consumption data only to the homeowner's phone (*unicast*), share water quality information with connected appliances (*multicast*), or broadcast a leakage alert to every device in the household network. Similarly, in industrial control systems, a programmable logic controller may multicast configuration parameters to a subset of actuators, broadcast an emergency stop

signal across the production line, or unicast diagnostic commands to a specific sensor. Because such messages directly influence physical processes and operational safety, secure key establishment becomes a foundational requirement.

However, existing group key establishment (GKE) mechanisms are not well suited for large-scale and dynamic IoT deployments, where devices are resource-constrained and group membership changes frequently. Centralized architectures such as hub-based fabrics [10] simplify initialization and coordination, but introduce scalability bottlenecks and single points of failure. Human-assisted pairing approaches [11,13,14] are practical only for small groups and become infeasible at IoT scale. Symmetric-key schemes based on a Key Distribution Center (KDC) [9] reduce local computation, but typically require costly system-wide rekeying whenever devices join or leave the network. Public-key approaches eliminate centralized key distribution, but many existing constructions incur substantial communication and computation overheads that are impractical for constrained IoT hardware.

This reveals a structural gap: a practical IoT key establishment protocol must support unicast, multicast, and broadcast communication within a single framework, operate without continuous reliance on a central controller, and allow efficient device addition and revocation through localized updates, while remaining feasible within the strict computation, memory, and energy budgets of low-power IoT devices.

To address this problem, we propose *TreeCast*, a scalable group key establishment protocol based on a binary tree of hashed Diffie–Hellman secrets. Devices occupy leaves of the tree and derive communication keys from secrets along their path to the root. Pairwise leaf secrets enable unicast communication, least-common-ancestor (LCA) secrets enable multicast, and the root secret enables broadcast. Because membership changes affect only a single leaf-to-root path, rekeying cost grows logarithmically with the group size while remaining localized.

Contribution: We introduce *TreeCast*, a group key establishment protocol tailored to device-to-device IoT communication. Our main contributions are:

1. **Communication-aware tree construction.** We design a binary tree of hashed Diffie–Hellman secrets that exposes intermediate subtree keys, naturally unifying unicast, multicast, and broadcast within a single hierarchical structure.
2. **Localized dynamic rekeying.** Membership changes update only a leaf-to-root path, yielding logarithmic rekey cost independent of total group size while preserving confidentiality of past and future communications.
3. **Security and asymptotic analysis.** We formalize the security model under standard hardness assumptions and derive computational, communication, and storage bounds showing predictable scalability.
4. **Implementation and evaluation.** We implement TreeCast on a 64 MHz Arm Cortex-M33 without hardware accelerators and demonstrate practical timing, energy, and memory overheads suitable for constrained IoT platforms.

Artifact Availability: The entire source code and artifact underlying this paper, including the implementation and benchmarkings, can be found at `https://github.com/KULeuven-COSIC/TreeCast`.

2 Related Work

Secure group communication in IoT and distributed systems has been studied across symmetric sensor-network schemes, tree-based continuous group key agreement, publish–subscribe encryption, contextual pairing, and hierarchical identity-based approaches. LEAP [20] establishes multiple symmetric key types per node to localize compromise impact and avoid public-key cryptography, but its multicast efficiency depends on physical neighbourhood clustering and rekeying cost scales with affected neighbours rather than logarithmically in total group size. TreeSync [19], the tree-based component of MLS, achieves forward secrecy and post-compromise security via a ratcheting tree in the Continuous Group Key Agreement model, yet assumes reliable ordered delivery, consistent global state, and comparatively capable devices, and derives a single evolving group secret rather than exposing intermediate subtree keys for communication-aware multicast. JEDI [6] leverages identity-based encryption and key regression for scalable publish–subscribe revocation, but targets cloud-backed systems, incurs ciphertext expansion proportional to revoked members, and does not provide bounded logarithmic per-device rekey updates for constrained microcontrollers. IOTCUPID [4], based on Partitioned-GPAKE [5], enables distributed group pairing for heterogeneous IoT devices, yet membership updates require re-execution or coordinated steps, limiting efficiency under frequent churn. Context-based pairing such as Perceptio [3] reduces explicit credential exchange but depends on environmental entropy and does not address scalable long-lived group maintenance. Hierarchical identity-based encryption, including T-HIBE [15] and WKD-IBE constructions [1], supports delegated key derivation but introduces ciphertext growth with hierarchy depth and costly revocation.In contrast, TreeCast treats intermediate subtree secrets as first-class communication primitives rather than deriving only a single evolving group key. This enables unicast, multicast, and broadcast within one unified hierarchy, while restricting membership updates to a single leaf-to-root path with logarithmic rekey cost independent of total group size or revoked-set size. Unlike prior work focused primarily on secure messaging or cloud-backed publish–subscribe systems, TreeCast is explicitly designed and experimentally evaluated for software-only execution on constrained Cortex-M class IoT devices.

3 TreeCast

TreeCast is a lightweight group key establishment protocol that organises IoT devices as leaves of a binary tree and derives communication keys using Hashed Diffie–Hellman (HDDH). Instead of maintaining only a single evolving group key, TreeCast exposes secrets at every internal node of the hierarchy. Leaf-level secrets

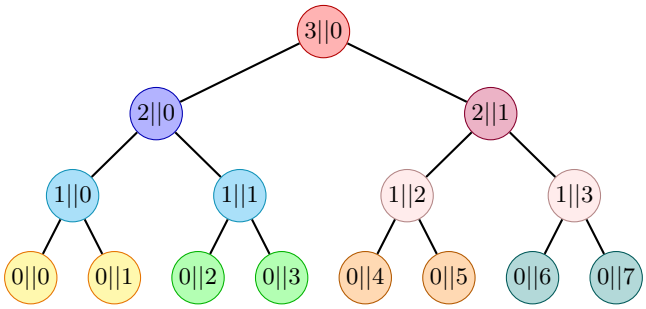

Fig. 1: Identities of the devices in the binary tree

enable pairwise unicast, least-common-ancestor secrets enable multicast within subtrees, and the root secret enables broadcast. This unified structure aligns with common device-to-device communication patterns while keeping cryptographic overhead predictable.

TreeCast is structurally inspired by the asynchronous ratcheting tree (ART) of Cohn-Gordon et al. [12], but differs in objective and mechanism. ART focuses on continuous group key agreement with strong post-compromise guarantees through stateful ratcheting and derives only a single root secret. TreeCast instead derives secrets at each internal node and avoids continuous ratcheting, trading maximal post-compromise security (PCS) guarantees for a lighter design tailored to constrained microcontrollers.

Formally, each device i holds a private key x_i and corresponding public key X_i. A device computes its parent secret by applying HDDH to its sibling's public key and its own private key. This derivation proceeds recursively up the tree until the root secret is obtained. After initial onboarding and public-key authentication via certification authorities, normal operation requires only local computation and symmetric encryption such as AES.

Dynamic membership is handled through localized rekeying. When a device joins or leaves, only the secrets along the affected leaf-to-root path are recomputed. This requires one hash evaluation per tree level and yields logarithmic rekey cost in the group size, while devices outside the path remain unaffected. As a result, TreeCast supports scalable and communication-aware key management without relying on a central controller during steady-state operation.

3.1 Identity Assignment

Each IoT device is first assigned a unique nonnegative integer identity by the certification authorities (CAs). This identity determines the device's position as a leaf in the TreeCast hierarchy. Internal node identities are then derived locally by the devices themselves without requiring interaction with the CAs. The CAs only verify the correctness of these assignments rather than generating them. This approach ensures that revoking a device does not require renumbering any other node in the tree. The identity assignment process is illustrated in Fig. 1.

Let x_{ij} denote the $(j+1)^{\text{th}}$ node at level $(i+1)$ of the tree, where level 0 corresponds to the leaves. If a device has identity $0n$ at the leaf level, it derives the identity of its parent node as

$$\text{ParentID}(x_{0n}) = 1 \,||\, \left\lfloor \frac{n}{2} \right\rfloor,$$

where $||$ denotes string concatenation. More generally, the identity of the parent of node x_{mn} at level m is

$$\text{ParentID}(x_{mn}) = (m+1) \,||\, \left\lfloor \frac{n}{2} \right\rfloor,$$

which assigns each internal node an identifier composed of its level index followed by the index of its position within that level. The use of $\lfloor n/2 \rfloor$ corresponds to the natural parent–child relationship in a binary tree and ensures stable identifiers across dynamic membership events.

3.2 Threat Model

We assume an adversary with complete control over the communication channel, including the ability to intercept, replay, modify, inject, or reorder protocol messages. Certification Authorities (CAs) are assumed to be *honest-but-curious*. They correctly perform certification but may attempt to infer auxiliary information from observed metadata. During initialization, devices transmit their public keys to the CA through a secure authenticated channel.

Each device is assumed to contain a trusted hardware component (e.g., TPM [16]) storing a manufacturer-certified attestation key. Devices prove possession of this key through a challenge–response protocol during onboarding, ensuring that only legitimate devices obtain valid certificates.

We consider both passive and active adversaries. Passive adversaries attempt to infer secret session or group keys from observed protocol transcripts, while active adversaries may impersonate devices or inject malicious protocol messages. TreeCast aims to provide forward secrecy, post-compromise security (PCS), authentication, replay and man-in-the-middle resistance, key robustness, key confirmation, and contributory key control. Our formal analysis focuses on the indistinguishability of derived secret keys under the HDDH assumption, while resistance against active attacks is achieved through authentication, freshness guarantees, and certificate-based identity verification.

3.3 TreeCast Protocol

Initialization The central authority announces the curve equation, the group generator, that is, the group parameters hash function, makes these parameters publicly available, and authenticates a set of certificate authorities (CAs). Each device i generates a Diffie–Hellman key pair $(g^{k_{0i}}, k_{0i})$ and a signing/verification pair $(\texttt{Sign}_{x_{0i}}, \texttt{Verf}_{x_{0i}})$, then sends $g^{k_{0i}}$ and $\texttt{Verf}_{x_{0i}}$ over a secure channel to

the CAs, which record them in some public ledger (and devices keep their private keys). Multiple CAs distribute trust. The CAs assign environment-specific unique IDs, publish the associated public keys and (e.g., ECDSA-signed) certificates, and devices retrieve public keys of other devices directly from the ledger. Devices then deterministically construct IDs for their parent and intermediate nodes; the CAs can verify group membership, and these IDs are then mapped to the leaf nodes of a binary tree, based on a required IoT environment. TreeCast operates fully distributed after initialization. Certification Authorities are used only to authenticate public keys initially, while all key derivation, communication, and rekeying proceed without central coordination.

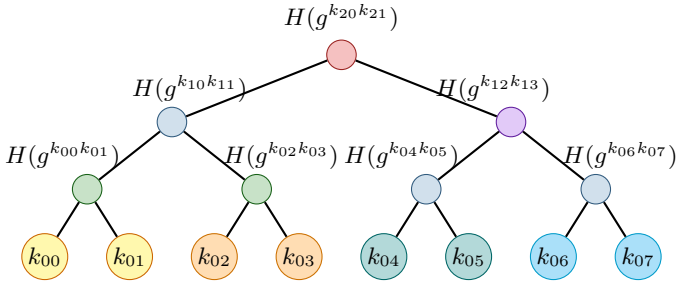

Fig. 2: Recursive key derivation in a binary tree structure (colour-enhanced for clarity)

Setup

1. Each device x first fetches the public key of $s(x)$ ($s(x)$ denotes the sibling node of x) from the ledger with the desired ID. If it has ID $0||n$ where n is even, then it will ask for the public key of the devices with identity $0||(n+1)$ and $0||(n-1)$ otherwise.
2. Each device x computes the secret key for $p(x)$ ($p(x)$ denotes the parent node of x) using the public key of $s(x)$ in the tree and its own secret key. Let a node with identity $m||n$ be denoted as x_{mn} as shown in Figure 1 and k_{ij} denote the secret key of x_{ij}. Let k_1 be the secret key of x_{0n}, and let g^{k_2} be the public key of its sibling node $x_{0(n+1)}$, if n is even, $x_{0(n-1)}$ otherwise.
3. The secret key of the parent node $x_{1(\lfloor n/2 \rfloor)}$ is computed as:

$$sk_{\text{parent}} = H\left((g^{k_2})^{k_1}\right) = H\left(g^{k_1 k_2}\right),$$

   where $H(\cdot)$ is a hash function from $\mathbb{G}$ to $\mathbb{Z}_q$, where q is a prime and $|\mathbb{G}| = q$.
4. The corresponding public key of the parent node becomes:

$$pk_{\text{parent}} = g^{sk_{\text{parent}}} = g^{H(g^{k_1 k_2})}.$$

5. Importantly, if a device x has no sibling node, then it selects $k \xleftarrow{R} \mathbb{Z}_q$, where $\xleftarrow{R}$ denotes uniform random sampling, and set it as secret key for its own and the secret key for $p(x)$, thus:

$$sk_{\text{parent}} = k$$

6. Each device computes its public key, submits it to the CAs for verification, whether the device is a leaf node of that subtree, rooted at that abstract node, and after approval, the key is uploaded to the ledger with the identity tag $x_{1(\lfloor n/2 \rfloor)}$.
7. The device then recursively retrieves the sibling public keys of its parent node from the ledger, derives higher-level keys (e.g., $x_{2(\lfloor \lfloor n/2 \rfloor /2 \rfloor)}$), and repeats the process until the root key is obtained (Fig. 2).

Session Key Establishment: Public-key operations are relatively expensive on resource-constrained devices, so we employ a hybrid encryption approach: a long-term public key exchange is used only to establish a short-term session key, and thereafter we will use a symmetric cipher (e.g., AES-128) to encrypt the actual messages.

Without Perfect Forward Secrecy (PFS): Let A and B hold long-term secret keys α, β with public keys g^α, g^β, and let gk denote the group root key. We will use the Key Derivation Function (KDF) [2] for generation of session keys. For round $i \in \mathbb{Z}_{\geq 0}$, the session key is derived as

$$K^i_{\text{session}} = \text{KDF}\big((g^{\alpha\beta} \text{ or } gk) \,\|\, i\big).$$

If an adversary compromises A or B and learns α or β, they can recompute $g^{\alpha\beta}$ and recover all previous session keys.

Unicast communication:

In unicast communication, one device will retrieve the long-term public key of the receiver's device and use it to generate the session key using fresh ephemeral keys generated by both devices.

- Let a and b be the long-term secret keys of parties A and B, respectively (with public keys g^a and g^b).
- In each new session, A samples a fresh ephemeral exponent x, computes, and sends g^x to B after signing on it.
- Likewise, B samples a fresh ephemeral exponent y, computes, and sends g^y to A after signing on it as well.
- Then both parties derive the shared session key as

$$K_{\text{session}} \;=\; H_1\big(g^{ab} \,\|\, g^{xb} \,\|\, g^{ay} \,\|\, g^{xy}\big)\,.$$

Multicast communication:

Multicast communication is performed through subtree keys associated with the LCA of the recipient set. To improve multicast efficiency, devices that communicate frequently can be placed within nearby subtrees so that smaller LCA

subtrees are used during key derivation. A sender retrieves the long-term public key of the corresponding LCA node and combines it with its own ephemeral secret to derive the session key, while the receiver nodes verify the signature on the sender's ephemeral public key. In addition, in broadcast communication, the root key will be used to generate the session key. Membership updates affect only the leaf-to-root path of the modified member, rather than the entire tree.

- Let a denote the long-term secret key of the sender device A. Let B denote a target set of leaf nodes (IoT devices) in the TreeCast hierarchy, corresponding to either a multicast subgroup or the entire network. Let b denote the secret key associated with the LCA subtree node covering B. The corresponding public keys are g^a and g^b, respectively.
- In each new session, A samples a fresh ephemeral exponent x, computes, and sends g^x to B, after signing on it.
- Then both parties derive the shared session key as

Key Update $$K_{\text{session}} \;=\; H_1\big(g^{ab} \,\|\, g^{xb}\big)\,.$$

1. **Update Trigger:** A periodic or event-driven key update is initiated for leaf node x_{06} in Figure 3.
2. **Leaf Key Regeneration:** Node x_{06} generates a fresh pair of private–public keys $\big(k'_{06},\, g^{k'_{06}}\big)$.
3. **Path Rekeying:** Node x_{06} recomputes all intermediate secret keys and corresponding public keys along the $path(x_{06})$ where $path(x)$ ancestor path from node x to the root node, and they are sent to the CAs, which upload them to the ledger.
4. **Distribution of New Keys:** Node x_{06} encrypts each new secret key under the public key of the nodes in $copath(x_{06})$, where $copath(x)$ denotes a set of siblings of nodes in $path(x)$ excluding the nodes in $path(x)$, and multicasts these secret keys to existing devices.
5. **Verification:** All recipients retrieve the updated public keys from the ledger, decrypt their shares, and verify the consistency of the rekey operation.

Revocation

1. **Revocation Trigger:** A device is revoked from the system, here x_{37} as shown in Figure 3. (e.g., due to compromise).
2. **Certificate Revocation:** The CAs revoke the certificate of the revoked device x_{07} and update the list on the ledger.
3. **Sibling Key Update:** The sibling node x_{06} of the revoked leaf node performs the Key Update method described above.

Addition The device addition procedure is similar to the revocation method.

3.4 Computational and Space Cost

Let n be the total number of devices and $h = \lceil \log_2 n \rceil$ the height of the binary tree (assuming $n = 2^m$ for simplicity).

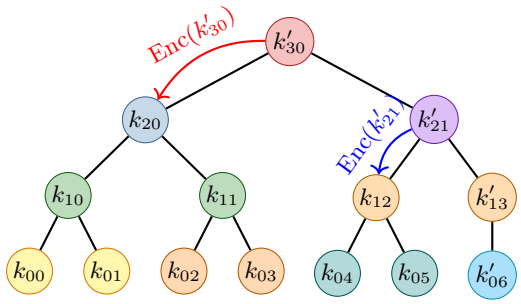

Fig. 3: Updated Tree after Revocation of x_{07} and Re-Keying by x_{06}

Per-Device Computational Cost Each device computes its own public key and signature key and derives the parent keys up to the root. The asymptotic operation counts are summarized below:

Operation	Per Device
Exponentiations	$2 + 2h = 2 + 2\lceil \log_2 n \rceil$
Hash evaluations	$h = \lceil \log_2 n \rceil$

Root Computation Time Since all devices at a given level operate in parallel, the total time to compute the root key is dominated by the tree height:

$$T_{\text{root}} \;=\; hT_H + (2h+2)T_e$$

where T_H and T_e are the costs of a hash evaluation and an exponentiation, respectively.

Space Requirement Each device stores secret keys along its path to the root, i.e., $\log_2 n$ keys. Assuming 128-bit keys:

$$\text{Storage per device} \;=\; 128 \log_2 n \text{ bits} \;\approx\; 0.128 \log_2 n \text{ KB}.$$

For example, a Wio Terminal [17] with 192 KB SRAM can easily support this storage overhead.

4 Implementation and Benchmarking

This section presents the complete implementation of TreeCast on a low cost microcontroller platform and reports the performance of its core cryptographic operations, communication paths and rekeying behaviour. All benchmarks were obtained on physical hardware using cycle accurate counters to provide reliable timing for short operations. The objective is to assess whether TreeCast is practical for constrained IoT devices that rely entirely on software based cryptography.

4.1 Testbed and Experimental Setup

All experiments were carried out on the Nordic nRF5340 Development Kit using the 64 MHz Arm Cortex-M33 application core, configured without TrustZone and without any hardware cryptographic accelerators. All hardware security blocks were explicitly disabled so that every primitive executes purely in software. This models the lowest common denominator for IoT platforms and provides a conservative lower bound for systems where optional accelerators may be present.

The on chip hardware timer runs at 32.768 kHz, which is insufficient for short operations such as hashing or memory accesses. All measurements therefore rely on the CPU cycle counter through `k_cycle_get_32()`, with time conversion performed using integer arithmetic at the fixed CPU frequency of 64 MHz.

4.2 Cryptographic Primitive Performance

Table 1 summarises the performance of the core primitives. The dominant cost is elliptic curve Diffie–Hellman, which requires roughly 1.47 million CPU cycles, corresponding to approximately 23 ms. SHA–256 hashing and AES–CTR encryption complete in microseconds. These values align with the performance expected from highly optimised software implementations on Cortex M class devices. Using a conservative current estimate of 10 mA at 3 V, the active power is approximately 30 mW. The energy consumption is computed as $E = P \cdot t$. The HDDH operation consumes approximately 0.69 mJ, while AES–CTR over 64 bytes costs around 3.7 μJ. Table 1 lists the corresponding energy and operations per Joule.

Table 1: Primitive performance and energy cost on the Cortex-M33)

Operation	Cycles (per op)	Time (ms)	Throughput (ops/s)	Energy	Ops per Joule (ops/J)
KeyGen (P-256)	1,466,796	22.918	44	0.69 mJ	1,450
Hash (SHA-256, 64 B)	5,859	0.091	11,000	2.7 μJ	370,000
HDDH (ECDH + hash)	1,470,703	22.979	43	0.69 mJ	1,440
AES–CTR (64 B)	7,812	0.122	8,200	3.7 μJ	270,000

4.3 Communication Modes

Table 2 presents the cost of unicast, multicast and broadcast communication. A unicast operation requires a fresh HDDH derivation and therefore inherits the cost of elliptic curve arithmetic. Multicast and broadcast operations require less than 200 cycles, which falls below the resolution of the 32.768 kHz hardware timer. This confirms that group communication becomes essentially free once the tree is established.

Table 2: Communication mode performance

Mode	Cycles (per op)	Time (ms)	Explanation
Unicast HDDH	1,470,703	22.979	fresh pairwise key
Unicast AES–CTR	7,812	0.122	payload encryption
Multicast retrieval	< 200	< 0.004	subtree lookup only
Broadcast retrieval	< 200	< 0.004	root lookup only

4.4 Dynamic Membership and Rekeying

Membership changes require updates only along the path from the modified leaf to the root. For a group of eight devices this corresponds to three SHA evaluations. The rekeying cost grows logarithmically with the group size because each additional level contributes a single hash computation. Table 3 reports the projected rekeying cost for different group sizes.

Table 3: Projected TreeCast rekeying cost based on measured SHA performance

Devices N	Height	Rekey cycles	Rekey time (ms)
10	4	12	0.366
100	7	21	0.640
1,000	10	30	0.915
10,000	14	42	1.281
100,000	17	51	1.556

Even for 100,000 devices the model predicts a full path rekey in under two milliseconds. This is more than an order of magnitude cheaper than a single unicast HDDH derivation and demonstrates that TreeCast scales efficiently to very large IoT deployments.

5 Evaluation

The evaluation confirms that TreeCast achieves practical performance on a constrained 64 MHz Cortex-M33 device using only software based cryptography. We assess unicast performance, group communication behaviour, rekey scalability and memory usage.

Unicast Performance: A full P-256 HDDH derivation completes in approximately 23 ms, which corresponds to around 40 pairwise key establishments per second. This rate is sufficient for realistic onboarding and device association workflows involving simultaneous joins or burst registrations. In practical deployments, network latency, radio contention and application logic dominate overall delay, making the cryptographic cost a small fraction of the total.

Group Communication Performance: Once the tree is initialised, multicast and broadcast operations require only memory access. Their execution time falls below the resolution of the hardware timer and is effectively negligible. This demonstrates a key benefit of TreeCast: the cost of addressing groups of any size is constant once the tree has been established. Symmetric encryption of a 64 byte payload requires approximately 0.12 ms and is independent of the number of devices.

Rekey Scalability: Rekeying affects only a single path in the tree and therefore grows logarithmically in the group size. As shown in Fig. 4, for 100,000 devices the projected cost is under 2 ms, compared to approximately 23 ms for a single unicast HDDH operation. This provides a compelling advantage over pairwise update approaches, particularly in deployments where membership changes are frequent.

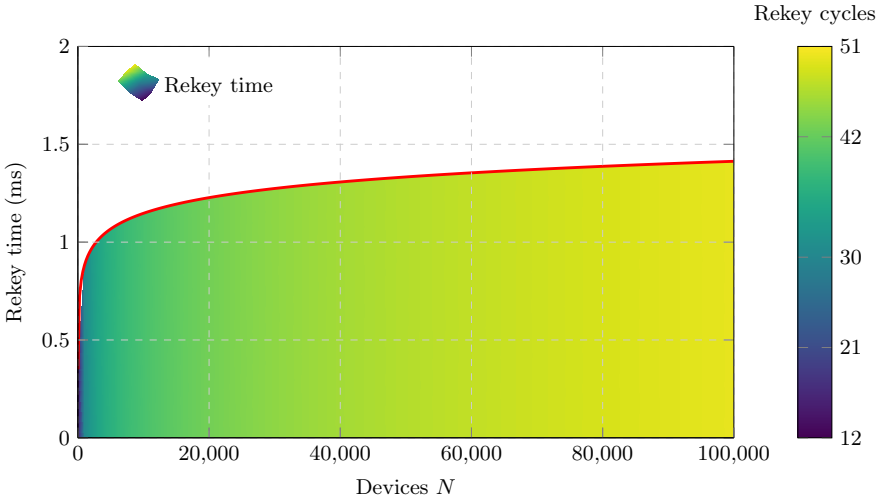

Fig. 4: Scalability of TreeCast rekeying on a Cortex-M33 microcontroller. Rekeying remains below 1.5 ms for 100,000 devices. The colour gradient shows the corresponding cost in CPU cycles.

Memory Usage: For a group of eight devices TreeCast requires 5188 bytes of RAM, dominated by stored secrets and keypairs. Node and keypair structures grow linearly with N while the cost of rekeying grows logarithmically. The memory footprint therefore remains moderate even for significantly larger groups and is well within the limits of low cost IoT microcontrollers.

Key takeaways: The implementation results confirm that TreeCast provides secure and scalable group communication on constrained microcontrollers using only software-based cryptography. Unicast key establishment completes in approximately 23 ms. Multicast and broadcast operations are effectively free once

the tree is initialised. Rekeying scales logarithmically and remains under 2 ms for networks of up to 100,000 devices. Energy costs are dominated by elliptic curve operations, with symmetric encryption in the few microjoule range. The overall memory footprint remains modest. These results demonstrate that TreeCast is a practical, efficient and deployable solution for large scale IoT systems.

Table 4: Comparative Analysis of Protocols Across Key Security Properties

Protocol	P_1	P_2	P_3	P_4
JEDI [6]	✓	✓	✗	✓
IOTCUPID [4]	✗	✗	✓	✓
Perceptio [3]	✗	✗	✗	✓
T-HIBE [15]	✓	✗	✓	✓
Symmetric Key [9]	✓	✗	✓	✗
This paper [TreeCast]	✓	✓	✓	✓

Protocol	P_5	P_6	P_7
JEDI [6]	✙	✗	✗
IOTCUPID [4]	-	-	✓
Perceptio [3]	-	✓	✓
T-HIBE [15]	✗	✓	✓
Symmetric Key [9]	✗	✓	✓
This paper [TreeCast]	◐	✓	✓

Property Assignments (Left Table):

P_1 = Scalability,
P_2 = Efficient Immediate Revocation,
P_3 = Avoid single TTP,
P_4 = Distributed.

Property Assignments (Right Table):

P_5 = Perfect Forward Secrecy (PFS),
P_6 = Post-Compromise Security,
P_7 = Efficiency for Constrained Devices,

Symbols for Protocol Properties: ✓ Fully supports ✗ Does not support ◐Partially supports ✙ Can be extended

Comparison with other works: As outlined in Section 2, existing group key establishment protocols span symmetric-key, identity-based and context-driven designs, yet none simultaneously satisfy scalability, dynamic membership, decentralization, and efficiency on constrained IoT nodes. Table 4 contrasts representative schemes against TreeCast across eight security and performance properties. Compared to JEDI, which incurs ∼3.83 s pairing and ∼6.50 s WKD-IBE encryption on Cortex-M0+ with ∼6.5 KiB RAM and ∼45 KiB metadata, TreeCast completes P-256 ECDH in ∼22.9 ms with microsecond hashing using < 6 KB state and $O(\log N)$ rekeying (< 2 ms at N=100k), delivering over 100× lower delay and better scalability; likewise, unlike IOTCUPID (∼39 ms for 20 devices on Cortex-A72)[4], TreeCast operates efficiently on low-power Cortex-M platforms without external sensing pipelines. Other systems reveal partial trade-offs: T-HIBE [15] scales but is computationally heavy; context-driven pairing like Perceptio [3] and IOTCUPID [4] remove gateways but rely on environmental stimuli; symmetric KDC-based methods [9] minimize local cost yet incur high join/leave rekey traffic. By contrast, TreeCast combines efficient re-distributed operation, logarithmic rekeying, immediate revocation, and strong cryptographic guarantees (authentication, key confirmation, post-compromise security, and partial forward secrecy), making it practical for large-scale heterogeneous IoT environments that rely solely on software-based cryptography.

6 Security Analysis

In this section, we present a comprehensive analysis of the TreeCast protocol, focusing on its resilience against both passive adversaries and a range of active attacks, including man-in-the-middle (MitM), replay, and known-key attacks.

6.1 Formal Key Secrecy Under the HDDH Assumption

We begin with resistance against the passive adversary, where an adversary can only eavesdrop on network traffic. Even with full access to all public parameters and all public keys of the internal and root node from the ledger, such an adversary cannot recover any of the secret keys.
Let $\mathbb{G}$ be a cyclic group of prime order q generated by g.
Let $\mathcal{B}_h$ be the set of all perfect binary trees of height h and $B \xleftarrow{R} \mathcal{B}_h$, with $K \xleftarrow{R} (\mathbb{Z}_q^*)^n$ and $n \leq 2^h$.

$$K = (k_{01}, k_{02}, \ldots, k_{0n}), \quad k_{0i} \xleftarrow{R} \mathbb{Z}_q^*.$$

- Definition of the adversarial view:

$$\mathsf{view}(h, K, B) \coloneqq \left\{ g^{k_{ij}} \;\middle|\; (i,j) \text{ are index nodes} \right\}$$

- Definition of the group key:

$$gk = x_{h0} \coloneqq H(g^{x_{(h-1)0} x_{(h-1)1}})$$

Here, $\mathsf{view}(h, K, B)$ denotes all values visible to a passive adversary on the ledger, and x_{h0} is the secret key at the root node.

Definition 1 (Tree Key Establishment Experiment $\mathsf{TKE}^{\mathbf{eav}}_{\mathcal{A},\Pi}(n)$).

1. *Multiple parties, each holding 1^n, execute the protocol Π (TreeCast) . This results published in a ledger containing all the intermediate node public keys, and a public root key g^k output by each of the parties.*
2. *The adversary $\mathcal{A}$ is given $\mathsf{view}(h, K, B)$ and g^k and outputs k'.*
3. *The output of the experiment is defined as 1 if $k' = k$, and 0 otherwise. In the case $\mathsf{TKE}^{eav}_{\mathcal{A},\Pi}(n) = 1$, we say that $\mathcal{A}$ succeeds.*

Definition 2 (Security Against Eavesdroppers). *A Tree key-exchange protocol Π is said to be secure in the presence of an eavesdropping adversary if for every probabilistic polynomial-time (PPT) adversary $\mathcal{A}$ there exists a negligible function $\mathsf{negl}(\cdot)$ such that*

$$\Pr\left[\mathsf{TKE}^{eav}_{\mathcal{A},\Pi}(n) = 1\right] \leq \mathsf{negl}(n).$$

Definition 3 (XY Experiment). *Let $\mathcal{G}$ be a group-generation algorithm, and $H : \mathbb{G} \to \mathbb{Z}_q$ be a hash function.*

1. *Run* $\mathcal{G}(1^n)$ *to obtain* $(\mathbb{G}, q, g)$*, where* $\mathbb{G}$ *is a cyclic group of order* q*,* $|q| = n$*, and* g *is a generator of* $\mathbb{G}$*.*
2. *Sample* $x, y \xleftarrow{R} \mathbb{Z}_q$ *uniformly at random, and compute* g^x *and* g^y*.*
3. *Give the adversary* $\mathcal{A}$ *the tuple*

$$(\mathbb{G}, q, g, g^x, g^y, g^{H(g^{xy})}),$$

and let $\mathcal{A}$ *output a value* $a \in \mathbb{Z}_q$*.*
4. *The output of the experiment is defined as*

$$\mathsf{XY}_{\mathcal{A},\mathcal{G}}(n) = \begin{cases} 1 & \text{if } a = H(g^{xy}), \\ 0 & \text{otherwise.} \end{cases}$$

Definition 4. *We say that the XY problem is hard relative to* $\mathcal{G}$ *if, for every probabilistic polynomial-time (PPT) adversary* $\mathcal{A}$*, there exists a negligible function* negl *such that*

$$\mathrm{Adv}^{\mathrm{XY}}_{\mathcal{A},\mathcal{G}}(n) = \Pr[\,\mathsf{XY}_{\mathcal{A},\mathcal{G}}(n) = 1\,] \leq \mathsf{negl}(n).$$

Lemma 1 (Hardness of XY). *If the HDDH problem is hard relative to* $\mathcal{G}$ *and* H *is modeled as random oracle, then the XY problem is also hard relative to* $\mathcal{G}$*.*

Proof. Assume, toward contradiction, that there exists a PPT adversary $\mathcal{A}$ such that

$$\mathrm{Adv}^{\mathrm{XY}}_{\mathcal{A},\mathcal{G}}(n) \geq \varepsilon(n) + \tfrac{1}{q}$$

for some non-negligible function $\varepsilon(n)$.

We construct a distinguisher $\mathcal{D}$ for the HDDH experiment.

Construction of $\mathcal{D}$*.*

1. Receive $(\mathbb{G}, q, g, g^a, g^b, T)$ from the challenger.
2. Compute g^T and give $(\mathbb{G}, q, g, g^a, g^b, g^T)$ to $\mathcal{A}$.
3. If $\mathcal{A}$ outputs $T' = T$, then $\mathcal{D}$ outputs 1, else 0.

If $T = H(g^{ab})$, the tuple given to $\mathcal{A}$ is identically distributed to the real XY experiment. Hence,

$$\Pr[\mathcal{D} = 1 \mid T = H(g^{ab})] = \Pr[\mathsf{XY}_{\mathcal{A},\mathcal{G}}(n) = 1] \geq \varepsilon(n) + \tfrac{1}{q}.$$

If $T = r \xleftarrow{\$} \mathbb{Z}_q$, then g^T is uniform and independent of g^a, g^b. Any guess for T succeeds with probability exactly $1/q$:

$$\Pr[\mathcal{D} = 1 \mid T \xleftarrow{R} \mathbb{Z}_q] = \tfrac{1}{q}.$$

Thus,

$$\begin{aligned}\mathrm{Adv}^{\mathrm{HDDH}}_{\mathcal{G},\mathcal{D}}(n) &= \Big|\Pr[\mathcal{D}=1 \mid T = H(g^{ab})] \\ &\qquad - \Pr[\mathcal{D}=1 \mid T \xleftarrow{R} \mathbb{Z}_q]\Big| \\ &\geq \left(\varepsilon(n) + \tfrac{1}{q}\right) - \tfrac{1}{q} \\ &= \varepsilon(n).\end{aligned}$$

using the inequality, $|a+b| \geq ||a| - |b||$

Since $\varepsilon(n)$ is non-negligible, this contradicts the HDDH assumption. Therefore, the XY problem is hard relative to $\mathcal{G}$.

Theorem 1. *If the HDDH problem is computationally hard in $\mathbb{G}$, then no PPT adversary $\mathcal{A}$ can compute the value gk given the $\mathsf{view}(h, K, B)$. Formally, for every PPT adversary $\mathcal{A}$ there exists a negligible function $\mathsf{negl}(\cdot)$ such that*

$$\Pr\big[\mathsf{TKE}^{eav}_{\mathcal{A},\Pi}(n) = 1\big] \leq \mathsf{negl}(n).$$

Proof Sketch. The structure of the argument follows the approach of Barua et al. [8]. For a complete exposition, we refer the reader to their work.

6.2 Security Properties Under Active Adversaries

We now analyse TreeCast under an active adversary model in which an adversary may intercept, modify, replay, or inject messages.

Post-Compromise Recovery: If the long-term secret of a device is compromised, an adversary can derive secrets along the corresponding $path(x)$. Recovery is achieved by regenerating the device's key pair and recomputing all secrets on that path. Since updates are confined to a single leaf-to-root path, subsequent sessions are protected after rekeying. In unicast mode, the use of fresh ephemeral values further limits exposure of future session keys following compromise.

Forward Secrecy: TreeCast provides different levels of forward secrecy depending on the communication mode. In unicast communication, session keys are derived from fresh ephemeral values contributed by both parties, yielding perfect forward secrecy: compromise of long-term keys does not reveal past session keys. In multicast and broadcast communication, session keys incorporate a fresh ephemeral contribution from the sender while reusing subtree secrets derived from long-term keys. Consequently, TreeCast provides partial forward secrecy in these modes: compromise of a receiver's long-term key may expose its past multicast session keys, while compromise of the sender's key does not reveal previously established sessions. Achieving full forward secrecy for multicast would require independent ephemeral exchanges with each recipient, resulting in substantial communication and computation overhead for constrained IoT

devices. TreeCast therefore adopts a practical efficiency–security trade-off that improves over static group-key approaches while remaining scalable and deployable in resource-constrained environments.

Authentication and Integrity: All public keys are certified by trusted authorities and verified before use. Ephemeral keys used in session establishment are signed and validated by the recipients. These mechanisms prevent man-in-the-middle attacks on key derivation and ensure message origin authentication.

Replay Resistance and Key Freshness: Each session derives its key using fresh randomness or nonces within a key derivation function. As a result, replayed messages do not lead to key reuse, and session keys remain unique and unpredictable.

Authentication and Man-in-the-Middle Resistance: All long-term public keys are certified by trusted authorities and verified before use. During session establishment, ephemeral values are digitally signed and validated by recipients. As a result, adversarial attempts to inject, modify, or substitute public keys or ephemeral contributions are detected during signature verification, preventing man-in-the-middle attacks on both tree derivation and session establishment.

Replay Resistance and Key Robustness: Each session key is derived using fresh randomness, either through ephemeral exponents or nonces incorporated into a key derivation function. Consequently, replayed messages do not result in key reuse, and previously established session keys cannot be used to derive future ones. The use of a second-preimage-resistant hash and KDF further prevents known-key attacks, ensuring that exposure of one session key does not reveal related secrets.

Key Confirmation and Control: Session keys are jointly derived from contributions of participating devices, ensuring that no single party can unilaterally determine the resulting key. Verification of derived values implicitly confirms that communicating parties share the same session key.

Together, these properties show that TreeCast maintains confidentiality, authentication, and controlled recovery under active adversarial interference while preserving scalability on constrained devices.

7 Conclusion and Future Work

This work introduced *TreeCast*, a distributed group key establishment protocol designed for large-scale and resource-constrained IoT deployments. TreeCast unifies unicast, multicast, and broadcast within a single tree structure and eliminates reliance on a central gateway during normal operation. Dynamic membership is supported through localized leaf-to-root rekeying with logarithmic cost. Under standard hardness assumptions, the protocol provides confidentiality and authentication with key confirmation, supports post-compromise recovery, achieves partial forward secrecy for multicast, and perfect forward secrecy for unicast. To strengthen security toward Forward Secrecy in closed environments such as smart homes, the protocol can incorporate a context-based nonce derived from environmental signals, similar to the approach in IOTCUPID, and

use it in session key generation. Assuming an adversary has no physical access to the environment, compromise of a device's long-term secret would only reveal past session keys corresponding to previously observed nonces, thereby providing additional resilience toward FS.

A software-only implementation on an Arm Cortex-M33 demonstrates practicality: unicast key derivation requires approximately 23 ms, multicast and broadcast reduce to constant-time key retrieval after tree establishment, and path rekeying for an eight-device group takes 0.366 ms. These results confirm that TreeCast is efficient on commodity microcontrollers and scales effectively with group size.

Future work includes extending the protocol to achieve full perfect forward secrecy for multicast communication in scenarios such as smart homes, generalizing the design to asymmetric hierarchical tree structures (e.g., Room → Floor → Building) for smart buildings and cities, incorporating external anonymity through pseudo-identities, and removing certificate authorities by enabling direct blockchain-based public key registration with robust membership validation. Exploring post-quantum cryptographic extensions remain open research directions for Tree-based key generation . An additional direction for future work is to investigate the implementation of the protocol using a public blockchain-based PKI to enhance transparency, decentralization, and trust without relying on traditional certification authorities.

Acknowledgments. This work was supported by the Flemish Government through the Cybersecurity Research Program, grant number VOEWICS02, and by the European Union's Horizon Research and Innovation program under grant agreement No. 101119747 (TELEMETRY).

References

1. Abdalla, M., et al.: Generalized Key Delegation for Hierarchical Identity-Based Encryption. Cryptology ePrint Archive, Report 2007/221 (2007)
2. Krawczyk, H.: Cryptographic Extraction and Key Derivation: The HKDF Scheme. Cryptology ePrint Archive, Paper 2010/264 (2010). `https://eprint.iacr.org/2010/264`
3. Han, J., Chung, A.J., Sinha, M.K., Harishankar, M., Pan, S., Noh, H.Y., Zhang, P., Tague, P.: Do You Feel What I Hear? Enabling Autonomous IoT Device Pairing Using Different Sensor Types. IEEE Symposium on Security and Privacy (2018)
4. Farrukh, H., Ozmen, M.O., Ors, F.K., Celik, Z.B.: One Key to Rule Them All: Secure Group Pairing for Heterogeneous IoT Devices. IEEE Symposium on Security and Privacy (SP), pp. 1–15 (2023)
5. Fiore, D., Vasco, M.I.G., Soriente, C.: Partitioned Group Password-Based Authenticated Key Exchange. Cryptology ePrint Archive, Report 2017/141 (2017)
6. Kumar, S., Hu, Y., Andersen, M.P., Popa, R.A., Culler, D.E.: JEDI: Many-to-Many End-to-End Encryption and Key Delegation for IoT. 28th USENIX Security Symposium (2019)

7. Joux, A.: A One-Round Protocol for Tripartite Diffie–Hellman. Algorithmic Number Theory, LNCS, pp. 385–394 (2000)
8. Barua, R., Dutta, R., Sarkar, P.: Extending Joux's Protocol to Multi-Party Key Agreement. Cryptology ePrint Archive, Report 2003/062 (2003)
9. Luo, X., Yin, L., Li, C., Wang, C., Fang, F., Zhu, C., Tian, Z.: A Lightweight Privacy-Preserving Communication Protocol for Heterogeneous IoT Environment. IEEE Access 8 (2020)
10. Duttagupta, S., Kolozyan, A., Nicolas, G., Preneel, B., Singelee, D.: What's the Matter? An In-Depth Security Analysis of the Matter Protocol. Cryptology ePrint Archive, Report 2025/1268 (2025)
11. McCune, J.M., Perrig, A., Reiter, M.K.: Seeing-is-Believing: Using Camera Phones for Human-Verifiable Authentication. IEEE Symposium on Security and Privacy (2005)
12. Cohn-Gordon, K., Cremers, C., Garratt, L., Millican, J., Milner, K.: On End-to-End Encryption: Asynchronous Group Messaging with Strong Security Guarantees. Cryptology ePrint Archive, Report 2017/666 (2017)
13. Li, X., Zeng, Q., Luo, L., Luo, T.: T2Pair: Secure and Usable Pairing for Heterogeneous IoT Devices. ACM CCS (2020)
14. Zhang, T., Yi, X., Wang, R., Wang, Y., Yu, C., Lu, Y., Shi, Y.: Tap-to-Pair: Associating Wireless Devices with Synchronous Tapping. ACM IMWUT (2018)
15. Duttagupta, S., Singelée, D., Preneel, B.: T-HIBE: A Novel Key Establishment Solution for Decentralized, Multi-Tenant IoT Systems. IEEE CCNC (2022)
16. Microsoft Azure: TPM Attestation in Azure IoT DPS. `https://learn.microsoft.com/en-us/azure/iot-dps/concepts-tpm-attestation`
17. Seeed Studio: Get Started with Wio Terminal. `https://wiki.seeedstudio.com/Wio-Terminal-Getting-Started/`
18. Lu, L., Lu, J.: A Lightweight Verifiable Secret Sharing Scheme in IoTs. Cryptology ePrint Archive, Report 2022/395 (2022)
19. Wallez, T., Protzenko, J., Beurdouche, B., Bhargavan, K.: TreeSync: Authenticated Group Management for Messaging Layer Security. USENIX Security (2023)
20. Zhu, S., Setia, S., Jajodia, S.: LEAP: Efficient Security Mechanisms for Large-Scale Distributed Sensor Networks. ACM CCS (2003)

A Assumptions & Threat Model

A.1 Assumptions

Definition 5 (HDDH Experiment). *Let $\mathcal{G}$ be a group-generation algorithm. $H : \mathbb{G} \to \mathbb{Z}_q$ be a hash function. The Hashed Decisional Diffie–Hellman (HDDH) experiment $\mathsf{HDDH}_{\mathcal{A},\mathcal{G}}(1^\lambda)$ is defined as follows:*

1. *Run $\mathcal{G}(1^\lambda)$ to obtain $(\mathbb{G}, q, g)$.*
2. *Sample $a, b \xleftarrow{R} \mathbb{Z}_q$ and compute g^a and g^b.*

3. *Sample a bit* $b^\star \xleftarrow{R} \{0,1\}$.
   - *If* $b^\star = 1$, *set* $T = H(g^{ab})$.
   - *If* $b^\star = 0$, *choose* $T \xleftarrow{R} \mathbb{Z}_q$ *uniformly at random.*
4. *Give* $(\mathbb{G}, q, g, g^a, g^b, T)$ *to the adversary* $\mathcal{A}$.
5. *The adversary outputs a bit* $\hat{b}$.

The experiment outputs 1 *if* $\hat{b} = b^\star$, *and* 0 *otherwise.*

Now,

$$\mathsf{Adv}^{\mathsf{HDDH}}_{\mathcal{A}}(\lambda) = \Big|\ \Pr[\mathsf{HDDH}_{\mathcal{A},\mathcal{G}}(1^\lambda) = 1] - \tfrac{1}{2}\ \Big|.$$

The HDDH problem is hard if $\mathsf{Adv}^{\mathsf{HDDH}}_{\mathcal{A}}(\lambda)$ is negligible for all PPT adversaries $\mathcal{A}$.

We assume that all hash functions used in the protocol are modeled as random oracle and that the HDDH problem is hard relative to $\mathcal{G}$.

Theorem 2. *If the HDDH problem is computationally hard in* $\mathbb{G}$, *then no PPT adversary* $\mathcal{A}$ *can compute the value* gk *given the* $\mathsf{view}(h, K, B)$. *Formally, for every PPT adversary* $\mathcal{A}$ *there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that*

$$\Pr\big[\mathsf{TKE}^{eav}_{\mathcal{A},\Pi}(n) = 1\big] \leq \mathsf{negl}(n).$$

Proof Sketch. We will prove this using induction,

Base Case: $h = 1$ By the above lemma, we see that the HDDH problem reducible to the problem **XY** and we have assumed that HDDH is hard in $\mathbb{G}$, so there does not exist any PPT adversary who can solve **XY**.

Induction Hypothesis: Suppose, for some $h \geq 2$, there does not exist any PPT adversary who can compute the secret key of the root node $k_{(h-1)0}$ of a perfect binary tree of height given $\mathsf{view}(h-1, K, B)$.

Induction Step: We now show that no PPT adversary can compute gr from $\mathsf{view}(h, K, B)$.

Suppose that there exists a PPT adversary $\mathcal{B}$ who can compute gr from $\mathsf{view}(h, K, B)$. which implies either:

(a) the adversary $\mathcal{B}$ can solve the HDDH problem in $\mathbb{G}$, or
(b) the adversary $\mathcal{B}$ can compute $k_{(h-1)0}$ and $k_{(h-1)1}$ from $\mathsf{view}(h-1, K_1, B_1)$ and $\mathsf{view}(h-1, K_2, B_2)$.

Now we have assumed that (a) is hard and (b) is also hard by the induction hypothesis. Therefore, it follows that it is not possible to compute gk from $\mathsf{view}(h, K, B)$.

Remarks:

- We can extend our protocol to nested IoT environments (e.g., smart buildings or smart cities) by organizing devices hierarchically in a binary tree (Fig. 5). Devices sharing a physical location (e.g., the same room or floor) derive a common secret, enabling efficient key management across all layers.

B implementation

B.1 Software Architecture

TreeCast maintains a complete binary tree of group secrets. Each leaf stores a device specific secret derived from a locally generated keypair, while internal nodes store hashed Diffie–Hellman values derived from the two child nodes. A unicast operation requires a fresh HDDH computation between two device keypairs. Multicast and broadcast operations retrieve precomputed subtree or root secrets respectively and therefore require only memory access. A membership update modifies the path from the affected leaf to the root, requiring exactly one hash evaluation per tree level.

The implementation uses 5188 bytes of context state for a group of eight devices. Each internal node occupies 33 bytes and each P-256 keypair requires 96 bytes of storage. This memory footprint fits comfortably within the typical RAM budgets of Cortex-M33 class microcontrollers.

B.2 Benchmarking Methodology

Each primitive operation was executed 100 times. For each iteration we record C CPU cycles using `k_cycle_get_32()`, and compute the execution time

$$t(C) = \frac{C \cdot 10^9}{f_{\text{CPU}}},$$

with $f_{\text{CPU}} = 64\,\text{MHz}$. We report average cycles per operation, the corresponding time in milliseconds and the achieved throughput.

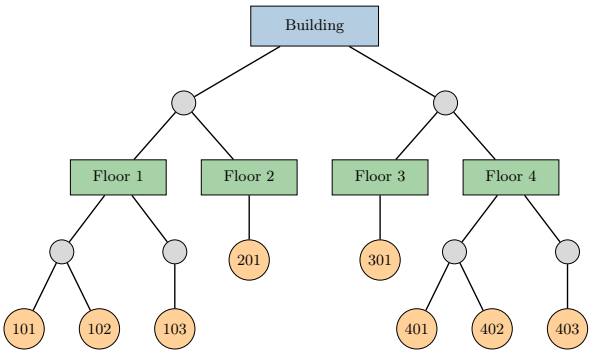

Fig. 5: Illustration of generalization of the Protocol in a nested environment