Multi-party Key Establishment for Resource-Constrained Devices

A Dissertation Submitted
in Partial Fulfilment of the Requirements for the Degree of

Master of Technology
in
Cryptology and Security

by

Supriyo Banerjee
[Roll No: CrS2325]

Under the Joint Supervision of

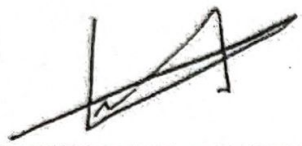
Prof. Dr. Bart Preneel
Prof. Dr. Mridul Nandi
Mr. Sayon Duttagupta

July 2025

This page intentionally left blank.

CERTIFICATE

This is to certify that the dissertation entitled *"Multi-party Key Establishment for Resource-Constrained Devices"* submitted by **Supriyo Banerjee** to **Indian Statistical Institute, Kolkata**, in partial fulfilment of the award of the degree of **Master of Technology in Cryptology and Security**, is a bona fide record of work carried out by him under my supervision and guidance. The dissertation has met all the requirements as per the regulations of this institute and, in my opinion, has reached the standard needed for submission.



Prof. Bart Preneel
Professor
COSIC
KU Leuven
Leuven, BELGIUM

Prof. Mridul Nandi
Professor
Applied Statistical Unit
Indian Statistical Institute
Kolkata-700108, INDIA

This page intentionally left blank.

Dedicated to all mothers...

ACKNOWLEDGEMENT

First and foremost, I would like to express my deepest gratitude to Prof. Bimal Kumar Roy, whose support and recommendation made it possible for me to come to COSIC, KU Leuven for my master's thesis. I will always be grateful for the opportunity and trust that he placed in me.

I would like to express my heartfelt gratitude to Prof. Bart Preneel and Prof. Mridul Nandi for giving me the opportunity to carry out my master's thesis work at COSIC, KU Leuven. Their support and encouragement have been a great source of motivation for me throughout this journey.

A special thanks to my daily supervisor, Mr. Sayon Duttagupta and also Dr. Dave Singelee for his constant guidance, constructive feedback, and for always being available to help me navigate technical challenges more effectively during this project.

I would like to express my sincere gratitude to Mr. Roozbeh Sarenche for his valuable suggestions and insightful ideas regarding the incorporation of blockchain-based PKI into my thesis.

I also would like to thank Pela Noe for her kind help and support during my stay in Leuven.

My sincere thanks to all my teachers at the Indian Statistical Institute for their teachings and advice, which laid the foundation for my research.

I am also very grateful to all my friends here in Leuven who came from ISI with me, as well as to Suparna Di, for their constant support and encouragement.

I am deeply grateful to my parents and family for their constant encouragement and unconditional support. I also thank my friends for always being there for me.

Lastly, to anyone who I may have missed, please accept my sincere thanks.

Supriyo Banerjee
Leuven
Belgium

ABSTRACT

As the number of IoT (Internet of Things) devices continues to grow, ensuring secure communication among them has become increasingly important. Traditional pairing schemes rely on centralized architectures, which are vulnerable to temporary or permanent failures due to operational malfunctions of their central hubs or gateways. To address these challenges, decentralized communication is essential.

However, existing decentralized pairing schemes suffer from high pairing times and significant computational overhead. Given the diverse capabilities of IoT devices, ranging from high-performance edge devices to resource-constrained sensors, many of these schemes become impractical in real-world scenarios. Therefore, we require a lightweight pairing scheme.

Our goal is to design a lightweight and decentralized group key establishment protocol that ensures strong security guarantees while remaining scalable and efficient. Our approach aims to reduce pairing time and computational complexity, making it suitable for a wide range of IoT applications, from smart homes to large-scale industrial networks.

Our approach builds upon concepts from existing works like Asynchronous Ratcheting Tree [9] and Extending Joux's Protocol to Multi Party Key Agreement [5], which focuses on decentralized, secure, and scalable group pairing scheme for dynamic environment.

Keywords: *Tree-based Group Key Establishment, Lightweight Cryptography, Blockchain-based PKI, IoT security, Decentralized Key Establishment. Multicast, Broadcast*

Contents

List of Figures

List of Tables

Notation

Symbol	Description
λ	Security parameter
$\mathcal{A}(1^\lambda)$	PPT adversary with input security parameter
$\xleftarrow{R}$	Uniform random sampling
$i_1(\cdot), i_2(\cdot)$	Hash functions: $i_1 : \mathbb{G} \to \mathbb{Z}_q$, $i_2 : \mathbb{G}_1 \to \mathbb{Z}_q$, assumed preimage resistant
$\|\|$	Concatenation
$\text{negl}(\lambda)$	Negligible function
$\text{Path}(x)$	Ancestor path from node x to root
$s(x), p(x)$	Sibling and parent of node x
$\text{Copath}(x)$	Set of siblings of nodes on $\text{Path}(x)$ excluding nodes on $\text{Path}(x)$
$\text{Enc}_K(M)$	Encryption of message M under key K
$E(\mathbb{F}_q)$	Elliptic curve over finite field $\mathbb{F}_q$
$\mathbb{G}_1, \mathbb{G}_2$	Additive and multiplicative groups, respectively
g	Generator of cyclic group $\mathbb{G}$
P_1	Generator of group $\mathbb{G}_1$
q	Prime order of $\mathbb{G}, \mathbb{G}_1, \mathbb{G}_2$
$\text{KDF}(\cdot), H(\cdot)$	Key Derivation and Hash functions
gk	Secret key of root node
T_{ι_1}, T_{ι_2}	Time to evaluate hash functions $i_1(\cdot), i_2(\cdot)$
T_e	Time for modular exponentiation or group operation (e.g. g^x, $(g^y)^x$)
T_p	Time to evaluate bilinear pairing
x_{ij}	$(j+1)^{\text{th}}$ node in $(i+1)^{\text{th}}$ level from leaf/base level
k_{ij}	Secret key of node x_{ij} (binary & ternary tree)

List of Abbreviations

CA	Certificate Authority
DH	Diffie–Hellman
EK	Ephemeral key
KDF	Key Derivation Function
MITM	Man-in-the-Middle Attack
ID	Identity or Identifier
IoT	Internet of Things
KDC	Key Derivation Center
GKE	Group Key Establishment
PFS	Perfect Forward Secrecy
PCS	Post-Compromise Security
PKI	Public Key Infrastructure
PPT	Probabilistic Polynomial-Time Algorithm
FS	Forward Secrecy
ECDSA	Elliptic Curve Digital Signature Algorithm
IBE	Identity-Based Encryption
PKG	Private Key Generator
JEDI	Joining Encryption and Delegation for IoT
GPAKE	Group Password Authenticated Key Exchange

Chapter 1

Introduction

As human civilization advances, we are becoming more dependent on machines, from washing machines to dish washers. With the advancement of technology, we now have smart versions of almost everything: smartwatches, smart thermostats, smart heaters, and many more. These devices often communicate with each other to work efficiently and autonomously. This is where cryptography comes into the picture as a hero, ensuring that their communication remains secure and trustworthy.

As the number of IoT devices grows rapidly [3], ensuring secure communication among them has become increasingly important, especially as these devices are deployed in sensitive environments such as smart homes, smart buildings, and also some public premises such as airports, railway stations, etc. Traditional group key establishment protocols, in many IoT systems, such as the current state-of-the-art Matter protocol [10], rely on centralized architectures, which are vulnerable to a single point of failure. To overcome these problems, we need a decentralized mechanism. However, most existing decentralized group key establishment protocols suffer from high computational costs and inefficient device addition and revocation method, which is not suitable for low-power, resource-constrained devices and large IoT environments.

Moreover, scalability becomes a major concern as the number of devices continues to increase, making it harder to efficiently manage secure communication. Furthermore, heterogeneity among IoT devices in terms of hardware capabilities, communication standards, and sensing modalities introduces additional complexities in designing a lightweight and uniform key establishment protocol for IoT systems.

1.1 Motivation

Although it may appear that the communicated messages are not important enough to require security, we will prove with a simple example that this is not the case. For example, imagine a smart door lock, if it is compromised remotely, an attacker could gain physical access to your home. This shows how weaknesses in GKE protocols can have serious real-world consequences.

As we discussed earlier, centralized protocols rely on trusted third parties or a central gateway/hub for secure communication. However, these entities can be a potential single point of failure and may not always be available, making centralized solutions less reliable and scalable in practice.

Several existing decentralized secure GKE protocols rely on human involvement, such as the use of cameras, 2D barcodes, synchronized shaking, and hand gestures [28]. Examples include protocols such as Tap2Pair [30] and T2Pair [22]. Although these approaches are effective for small-scale settings, they scale poorly in environments with a large number of devices.

Also, symmetric cryptographic systems are lightweight and efficient; they typically rely on a Key Distribution Center (KDC) for the pre-distribution of secret keys[27]. However, whenever a device is added or revoked, the KDC must generate a new group secret and securely distribute it to the remaining devices. This constant need for key renewal and secure redistribution limits scalability and makes symmetric approaches impractical for dynamic networks where devices frequently join or leave.

Furthermore, many public-key-based protocols, such as IOTCUPID [14] have an inefficient revocation method, and JEDI [21] uses IBE, which uses PKG, which is a trusted third party and is vulnerable to single of point failure. Furthermore, resource constraints in IoT devices such as limited space, computational power, and battery life make it impractical to directly apply complex cryptographic schemes without optimization.

To overcome these limitations, we propose a more robust approach based on public key cryptography. Despite its higher computational cost compared to symmetric schemes, it offers scalability.

In real-world IoT systems, devices are not guaranteed to be online at the same time, making synchronous protocols impractical. This motivates the need for an asynchronous group key establishment protocol.

The primary challenge is to design a protocol that maintains strong security guarantees while minimizing computational and communication overhead, making it truly practical for lightweight, resource-constrained IoT environments.

1.2 Research Problem

Our goal is to develop a secure and decentralized group key establishment protocol tailored for IoT systems. The key requirements of the proposed protocol are outlined below:

- **Decentralization:** The protocol should operate without relying on any central trusted authority.
- **Security and Authentication:** It must ensure that all communicating devices are authenticated and that communication remains secure against adversarial attacks.
- **Dynamic Membership Handling:** The protocol must support secure addition and revocation of devices, preserving the following:
  - Newly added devices must not access past communications.
  - Revoked devices must not access future communications.
- **Low Pairing Time and Computational Overhead:** The protocol should be lightweight enough for resource-constrained IoT devices to execute efficiently.
- **Scalability:** The protocol must scale well with the increasing number of devices in large deployments.
- **Asynchronous Communication:** Devices should be able to participate in the protocol without the need to be online simultaneously.
- **Efficiency:** Requires minimal computation and storage overhead, especially suitable for constrained devices.

Now we will discuss some more properties that we want in our protocol. A summary of the goals is provided in Table 1.1.

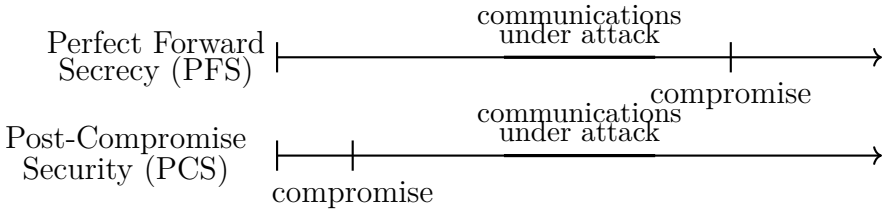


Figure 1.1: llustration of Forward Secrecy and Post-Compromise Security

Table 1.1: Protocol Requirements: Security Goals

Security Goal	**Description [6] [4]**
Perfect Forward Secrecy (PFS)	Guarantees that compromise of long-term keys does not affect past session keys.
Post-Compromise Security	Ensures that even if a device is compromised at a certain point in time, all future communications remain secure after a certain recovery period.
Resistant against Known-Key Attack	A protocol is said to be vulnerable to a known-key attack if compromise of past session keys allows either a passive adversary to compromise future session keys, or impersonation by an active adversary in the future.
Key Confirmation	Assures both parties possess the same session key, enhancing trust and preventing certain impersonation attacks.
Key Freshness	Ensures that the session key is newly generated and not reused, mitigating known-key attacks.
Key Control	Prevents either party from forcing the session key, ensuring joint contribution to key generation.
Mutual Entity Authentication	Ensures both parties verify each other's identity to prevent impersonation or spoofing in one-to-one communication.
Unilateral Entity Authentication	Ensures receivers to verify sender's identity to prevent impersonation or spoofing in one-to-many communication.
External Anonymity	Hides identities of the devices from external observers while allowing mutual recognition during the session.

1.3 Our Contribution

In this work, we propose three decentralized key establishment protocols that follow a hierarchical tree-based structure to support secure communication among IoT devices.

- **Protocol 1** is based on a **binary tree** structure the same as the asynchronous ratcheting tree [9]. Each pair of sibling nodes performs a basic Diffie-Hellman key exchange to generate a shared secret and assign it to an abstract parent node in a binary tree. The leaf nodes then generate the secret key of the parent node of the parent node. So, the secret key and public key pairs of the abstract internal node are generated by the leaf nodes. Eventually, they will generate the secret key of the root node for broadcasting. This approach ensures efficiency and simplicity while maintaining security, making it suitable for lightweight IoT environments.
- **Protocol 2** extends this idea to a **ternary tree**, where each internal node is connected to three children. The motivation behind this approach was to reduce the height of the tree. For key agreement among three nodes, we use **Joux's tripartite key exchange protocol** [19] based on pairing-based cryptography. This enables secure

shared secret generation among three devices simultaneously, reducing the overall tree height for a fixed number of devices. It turns out that Protocol 1 is always efficient in real circumstances, we will discuss it later in the Evaluation chapter.

So, Protocol 1 offers scalable and secure key establishment solution for a wide range of IoT network environments.

1.4 Thesis Outline

The remainder of this thesis is organized as follows.

- **Chapter 2** reviews the related literature and previous approaches to group key establishment among resource constrained devices, especially IoT device.
- **Chapter 3** introduces the fundamental concepts and cryptographic building blocks required to understand our work.
- **Chapter 4** presents in detail the proposed hierarchical key establishment protocols, including computational cost.
- **Chapter 5** offers a comprehensive security analysis, together with an evaluation of efficiency and suitability of the protocol for resource-constrained devices.
- **Chapter 6** we have compared our work with existing works in terms of security measures and computational costs.
- **Chapter 7** we have described the potential future research directions and concludes the thesis by summarizing the main contributions.

Chapter 2

Literature Reviewed

To understand the current state-of-the-art in secure key establishment for IoT systems, I have thoroughly reviewed several works in this domain. Among them, I highlight four prominent works that are highly relevant to my thesis. These selected works cover both centralized and decentralized approaches, offering a broad perspective on existing techniques. Notably, two of these works relies on a centralized architecture using symmetric key cryptography and verifiable secret sharing, while the others propose decentralized solutions based on public key cryptography, with a couple of them representing recent advancements in the field. The works are as follows.

1. **Lightweight Privacy-preserving Communication Protocol for Heterogeneous IoT Environment: [27]**

   This protocol [27] is for heterogeneous and resource-constrained IoT environments such as smart cities, vehicles, homes, etc. The protocol leverages symmetric key cryptography to minimize computational and energy costs. It uses AES-128 for encryption and SHA-256 for message authentication (MAC).
   Devices communicate with the control center (D2C) using persistent keys. For device-to-device (D2D) communication, the control center delegates a fresh session key using secure channels. All keys are updated synchronously to resist key reset and capture attacks.
   The protocol assumes a trusted control center. If this central server is compromised, all key material and control logic are at risk. Therefore, this protocol is vulnerable to single point of failure. If the key state of a device is compromised, past session keys derived via the Logistic Map are exposed. So, no support for forward secrecy beyond scheduled rekeying. The protocol is not scalable because each device has to store and stay perfectly in sync with its own chaotic key parameters. Also, since devices must contact the control center regularly to update their keys, the system does not work well if some devices go offline. As the number of devices grows, keeping everything in sync becomes harder and slower.

2. JEDI: [21]

JEDI [21] is a many-to-many end-to-end encryption protocol designed for IoT environments. JEDI uses WKD-IBE ([2]), a hierarchical identity-based encryption scheme that uses wildcard key derivations, which facilitates flexible and scalable key management.
JEDI achieves decentralized delegation, decoupling of communication parties, and end-to-end encryption optimized for resource-constrained devices. The scheme incorporates notable optimizations in WKD-IBE that make it suitable for such constrained environments.
In Identity-Based Encryption (IBE), a trusted third party, called the PKG, generates the corresponding private keys for the authorized parties who want to use their ID as their public key. Using the master private key, the PKG generates the private key for the identity ID. So, PKG acting as a centralized trusted third-party. Therefore, it also becomes vulnerable to single-point failure. In this protocol, the sender must know who is revoked before encryption. This weakens JEDI's 'sender-receiver decoupling' mechanism. Moreover, the length of ciphertexts grow with the number of revoked devices.

3. Perceptio: [16]

Perceptio [16] introduces a novel approach to context-based pairing that supports heterogeneous sensors by utilizing event timing as a common factor rather than relying on specific sensor data. The core idea is that devices located within the same physical boundary (e.g., a single-family home) observe overlapping event timings, which can be converted to event fingerprints for pairing. This timing-based approach enables devices with different sensing modalities to derive shared entropy for secure key establishment.
Despite its advantages, Perceptio has several limitations. It supports only sequential device addition and lacks concurrent or group keying capabilities. Furthermore, Perceptio requires sensor calibration to set appropriate thresholds for distinguishing signal from noise, which is critical given variations in sensor placement and environmental conditions. The assumption of physical boundary security limits applicability in public or shared spaces, although the protocol may extend to multi-tenant private buildings with proper threshold tuning. Finally, pairing time depends on activity frequency; homes with low activity may experience long pairing times unless supplemented by signal-injecting devices, which trade-off cost and usability for speed.

4. IOTCUPID: [14]

IOTCUPID [14] is a secure decentralized group pairing system designed for IoT devices equipped with heterogeneous sensing modalities. It uses Partitioned-GPAKE [16] and assumes physical proximity and shared environmental context among devices to establish cryptographic group keys without requiring active user involvement. IOTCUPID operates in three main phases: (a) detecting events sensed by both instantaneous and continuous sensors using a window-based derivation technique,

(b) grouping these events via a fuzzy clustering algorithm to extract inter-event timings as shared contextual evidence, and (c) establishing group keys among devices exhibiting identical inter-event timings through partitioned-GPAKE.
Despite these advantages, the revocation mechanism is highly inefficient, as it requires reinitialization of the entire protocol whenever devices need to be revoked or added, leading to reduced scalability. Moreover, since this protocol relies on the extraction of a shared context from real-world environmental events, it is not suitable for deployment in open environments. In such a scenario, an adversary could potentially exploit the same contextual information to join the group. These factors restrict its applicability primarily to small, closed environments with stable and shared physical contexts.

5. A Lightweight Verifiable Secret Sharing Scheme in IoTs: [25]

The proposed (t, n) Verifiable Secret Sharing (VSS) scheme uses Nyberg's One-Way Accumulator for one-way hash functions (NAHF) [29] to achieve lightweight and non-interactive share verification in IoT environments. The protocol significantly reduces communication costs by requiring only one verification value for all shares, making it highly efficient for low-power and resource-constrained devices. It also minimizes computation costs for both the dealer and participants, outperforming classical schemes such as Feldman [15].
Despite these advantages, the computational cost for the dealer increases linearly with the number of participants, leading to scalability challenges for large IoT networks. In addition, the scheme is vulnerable to a single point of failure due to its dependence on the dealer.

6. T-HIBE Protocol: [13]

T-HIBE (Threshold-based Hierarchical Identity-Based Encryption) is a cryptographic protocol designed to enable secure key establishment in decentralized, multi-tenant IoT environments. It builds upon the HIBE scheme by distributing the role of the root PKG among multiple non-colluding entities. This threshold-based approach mitigates the traditional key-escrow problem inherent in IBE/HIBE systems, where a single PKG holds the authority to generate all private keys and can thus decrypt any ciphertext. The T-HIBE protocol uses Shamir Secret Sharing to allow secure key derivation without a central trusted authority.
However, despite these advantages, T-HIBE has limitations: The ciphertext length grows linearly with the hierarchy depth, which increases computational overhead on constrained devices. Although temporary identifiers (ID || timestamp) can be used for revocation but there is no scope of immediate revocation. Frequent updates at higher levels require all children nodes to reinitialize, which is not scalable for dynamic, large-scale IoT networks. While the root PKG's power is decentralized, domain-level PKGs can still decrypt all communications within their subdomain.

Chapter 3

Cryptographic Background

In this chapter, we introduce the basic cryptographic concepts that form the foundation of our protocol.

3.1 Bilinear Maps

Let $\mathbb{G}_1$ and $\mathbb{G}_2$ be two finite additive groups of order q, $\mathbb{G}_T$ be a multiplicative cyclic group of prime order q. Let P_i be a generator of $\mathbb{G}_i$. A bilinear map is a function $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$. The bilinear map e has the following properties:

- Bilinearity: for all $R \in \mathbb{G}_1$ and $S \in \mathbb{G}_2$, $a, b \in \mathbb{Z}_q$, we have $e(aR, bS) = e(R, S)^{ab}$
- Non-degeneracy: e is non-degenerate, that is, $e(P_1, P_2) \neq 1$.

Furthermore, properties (1) and (2) imply that $e(R_1 + R_2, S) = e(R_1, S) \cdot e(R_2, S)$ for all $R_1, R_2 \in \mathbb{G}_1$, $S \in \mathbb{G}_2$.

If $\mathbb{G}_1 = \mathbb{G}_2 = \mathbb{G}_T$, the bilinear map is symmetric; otherwise, asymmetric. Unless otherwise mentioned, we consider only the bilinear maps that are efficiently computable and symmetric.

Proposition 3.1. *The bilinear map e is symmetric, that is, $\forall R, S \in \mathbb{G}_1$, $e(R, S) = e(S, R)$.*

Proof. Since $R, S \in \mathbb{G}_1$, $\exists a, b \in \mathbb{Z}_q$ such that $R = aP$ and $S = bP$. Thus,

$$e(aP, bP) = e(P, P)^{ab} = e(bP, aP)$$

Hence, $e(R, S) = e(S, R)$. □

3.2 Hash Function

A **hash function** is a function

$$H : A \to B,$$

where

$$A = \left\{a \in \{0,1\}^j : j \in \mathbb{N}\right\}$$

is the set of all bit sequences of arbitrary length, and

$$B = \{0,1\}^k$$

is the set of all bit sequences of a specific, generally short, length k. The inputs to the hash functions are called *messages*, and the outputs are called *digests*.

Definition 3.1. Second Preimage Resistant:

Second preimage resistant is a property of a hash function $H : A \to B$ which holds when, for any given $m \in A$, it is "hard" to find an $m' \in A$, $m' \neq m$, such that

$$H(m) = H(m').$$

We shall assume that all the hash functions used are second preimage-resistant.

3.3 ECDSA

The ECDSA signature scheme $(\mathsf{G}, \mathsf{S}, \mathsf{V})$ uses the group of points G of an elliptic curve over a finite field $\mathbb{F}_p$. Let g be a generator of G and let q be the (prime) order of the group G. We use multiplicative notation for the group operation. The scheme also requires a hash function $H : \mathcal{M} \to \mathbb{Z}_q^*$, where $\mathcal{M}$ is the message space.

Key Generation: $\mathsf{G}()$

- Choose $\alpha \leftarrow_R \mathbb{Z}_q^*$.
- Compute $u \leftarrow g^\alpha \in G$.
- Output the secret key $\mathsf{sk} := \alpha$ and the public key $\mathsf{pk} := u$.

Signing: $\mathsf{S}(\mathsf{sk}, m)$

To sign a message $m \in \mathcal{M}$ using secret key $\mathsf{sk} = \alpha$:

- Repeat:
  - Choose $\alpha_t \leftarrow_R \mathbb{Z}_q^*$.
  - Compute $u_t \leftarrow g^{\alpha_t} \in G$.
  - Let $u_t = (x, y) \in G$, where $x, y \in \mathbb{F}_p$.
  - Interpret x as an integer in $[0, p)$ and compute $r \leftarrow [x]_q \in \mathbb{Z}_q$ (i.e., reduce x modulo q).

  - Compute $s \leftarrow (H(m) + r\alpha) \cdot \alpha_t^{-1} \mod q$.
- Until $r \neq 0$ and $s \neq 0$.
- Output signature $\sigma := (r, s) \in \mathbb{Z}_q^2$.

Verification: $\mathsf{V}(\mathsf{pk}, m, \sigma)$

To verify a signature $\sigma = (r, s) \in \mathbb{Z}_q^2$ on a message $m \in \mathcal{M}$ using public key $\mathsf{pk} = u \in G$:

- If $r = 0$ or $s = 0$, output `reject` and stop.
- Compute $a \leftarrow H(m)/s \mod q$ and $b \leftarrow r/s \mod q$.
- Compute $\hat{u}_t \leftarrow g^a u^b \in G$.
- If $\hat{u}_t$ is the point at infinity, output `reject` and stop.
- Let $\hat{u}_t = (\hat{x}, \hat{y}) \in G$.
- Interpret $\hat{x}$ as an integer in $[0, p)$ and compute $\hat{r} \leftarrow [\hat{x}]_q$.
- If $r = \hat{r}$, output `accept`; else, output `reject`.

3.4 Key Derivation Functions

In this section, we formalize the notion of a Key Derivation Function (KDF) along with the concept of a "source of keying material." These definitions are crucial for reasoning about the security of key derivation in cryptographic protocols.

Definition 3.2 (**Key Derivation Function**). *A Key Derivation Function (KDF) is an algorithm that takes as input four arguments:*

- σ*: a value sampled from a source of keying material* Σ *(see Definition 3.3),*
- ℓ*: a desired output length in bits,*
- r*: a salt value sampled from a predefined domain (optional),*
- c*: a context string (optional),*

and outputs a string of ℓ *bits. The salt* r *and the context* c *can be set to the null string or to a fixed constant if not used.*

Definition 3.3 (**Source of Keying Material**). *A source of keying material* Σ *is defined as a two-valued probability distribution that outputs a pair* (σ, α)*, where:*

- σ *is the secret input to the KDF,*
- α *is auxiliary information correlated with* σ *and available to the adversary.*

The distribution is generated by an efficient probabilistic algorithm. The definition abstracts the input to Σ *but requires that* Σ *be efficiently samplable.*

3.5 Important Definitions:

Definition 3.4. n-Multilinear Map

We say that a map

$$e : G_1^n \to G_2$$

*is an n-**multilinear map** [7] if it satisfies the following properties:*

1. $\mathbb{G}_1$ *and* $\mathbb{G}_2$ *are groups of the same prime order;*
2. *For all* $a_1, \ldots, a_n \in \mathbb{Z}$ *and* $Q_1, \ldots, Q_n \in G_1$*, we have*

$$e(a_1 Q_1, \ldots, a_n Q_n) = e(Q_1, \ldots, Q_n)^{a_1 \cdots a_n};$$

3. *The map e is **non-degenerate** in the following sense: if* $P_1 \in G_1$ *is a generator of* G_1*, then*

$$e(P_1, \ldots, P_1)$$

is a generator of G_2*.*

Definition 3.5. Unicast

Unicast refers to one-to-one communication in which a message is sent from one sender to exactly one receiver.

Definition 3.6. Broadcast

Broadcast refers to one-to-all communication in which a message is sent from one sender to all devices or nodes within the network.

Definition 3.7. Multicast

Multicast refers to one-to-many communication in which a message is sent from one sender to a specific group of devices within the network, rather than to all devices.

3.6 Assumptions

Definition 3.8. Hashed Decisional Diffie-Hellman (HDDH) Assumption: *[1]*

The Hashed Decisional Diffie-Hellman (HDDH) assumption states that no efficient PPT algorithm can distinguish between the following two distributions with non-negligible advantage:

- $(q, g, g^a, g^b, H(g^{ab}))$*, and*
- (q, g, g^a, g^b, r)

where $a, b, r \xleftarrow{R} \mathbb{Z}_q$*.*

Formally, For every PPT, 0/1-valued algorithm $\mathcal{A}$*, the HDDH assumption is defined as:*

$$Adv_{\mathcal{A}}^{HDDH} = \left| \Pr\left[\mathcal{A}(q, g, g^a, g^b, H(g^{ab})) = 1\right] - \Pr\left[\mathcal{A}(q, g, g^a, g^b, r) = 1\right] \right| < negl(\lambda)$$

Chapter 4

Proposed Scheme

In this chapter, we present two key establishment protocols designed for secure communication in IoT systems. As discussed earlier, the two protocols are structurally similar, both relying on a tree-based approach for key generation and management. The first protocol uses a binary tree, while the second extends this idea to a ternary tree leveraging tripartite Diffie-Hellman key exchange. Our initial goal was to develop a generalized version of this approach by exploring the possibility of using an n-ary or asymmetric tree structure, where devices that often communicate with each other would naturally be placed as child nodes under the same abstract parent node. However, due to certain limitations, which we will discuss later in the remarks of this chapter and in the Future Work section 7.1, we were unable to fully realize this generalization. Thus, we consider **Protocol 1** as the main protocol in our proposed scheme.

As a result, we focus on presenting the binary and ternary tree-based protocols, both designed for the same set of use cases and threat model. The two protocols are as follows.

4.1 Protocol 1

This protocol builds upon the asynchronous ratcheting tree scheme described by Gordon et al. [25], with the addition of a blockchain-based PKI to manage public keys and public key certificates along with their identities, securely and in a decentralized way, ensuring tamper proof updates and verifiability.

All IoT devices are organized as leaf nodes in a binary tree. Each device (leaf nodes) independently computes the secret key for its parent node using the Hashed Diffie-Hellman key exchange with the public key of its sibling node. The corresponding public keys are uploaded to a distributed set of CAs, avoiding reliance on a single trusted party and enabling a decentralized PKI.

This recursive, bottom-up process continues until the root node's secret key is derived, which serves as the group key for broadcasting messages across the entire network. The public keys of intermediate nodes will be used for multicast communication, while their

secret keys act as shared symmetric keys for the leaf nodes of the subtree rooted at that node. This key establishment is a one-time setup. Subsequent updates, such as device revocation or addition, do not require repeating the entire process.

Given the computational overhead of public key encryption, we introduce a lightweight session key generation step. Public key methods are only used to establish short-term session keys, which will be treated as symmetric keys (e.g., via AES-128).

This protocol is well-suited for large-scale open environments such as airports, railway stations, or industrial facilities, where the communication channel is exposed to potential adversaries. In such scenarios, when the same message needs to be sent to multiple devices, intermediate nodes can be used for efficient multicast communication.

Although this protocol can also be applied in small closed environments like smart homes, it can become expensive due to the reliance on a blockchain-based PKI.

4.1.1 Threat Model:

CAs are assumed to be honest but curious. The devices will share the public keys to the CAs via a secure channel. Each device is equipped with a secure hardware chip (like a TPM) that holds a secret attestation key, which the device manufacturer certifies. During initialization, the device proves that it owns this key using a challenge-response method, ensuring that only genuine devices get certificates. So, the trust also lies with the device manufacturer.

An adversary is assumed to have complete knowledge of the protocol and complete access to the communication channel. This includes the ability to intercept, replay, modify, or delete messages transmitted over the network. A malicious device may generate arbitrary secret keys and send them to the sibling nodes during key updates in an attempt to compromise the group key.

Furthermore, adversaries may attempt to impersonate legitimate devices and send incorrect or misleading messages to disrupt the system. This makes the protocol vulnerable to potential attacks such as eavesdropping, replay attacks, known-key attacks, and man-in-the-middle attacks.

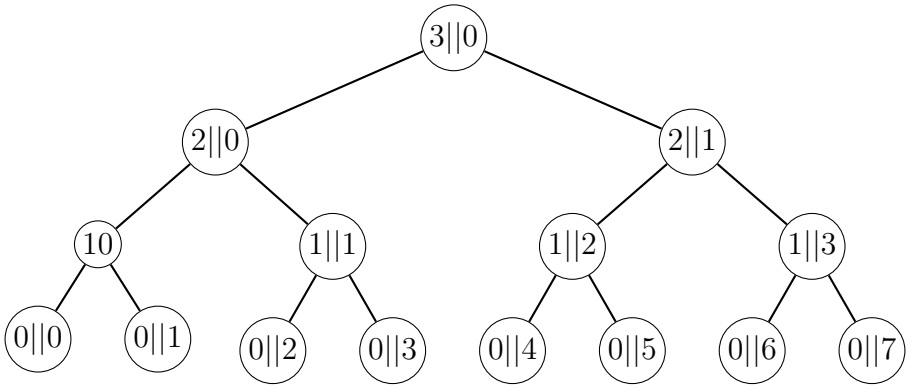


Figure 4.1: Identities of the devices in the binary tree

4.1.2 Identity Assignment:

First, the CAs assign a non-negative integer number as an identity for an IoT device. Subsequently, the devices assign IDs to their respective parent nodes independently of the CAs. The CAs will just approve it. With this identity assignment, we can revoke a device without affecting the IDs of the remaining nodes.

If the ID of a device is $0n$, then it assigns the ID of its parent node as:

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

So, in general, we have the follwing.

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

4.1.3 Protocol

I. Intialization:

The central authority announces the curve equation, the group generator, that is, the group parameters hash function, and makes these parameters publicly available. The CAs will be authenticated by the central authority. The devices will generate a pair of keys $(g^{k_{0i}}, k_{0i})$ and $(\texttt{Sign}_{x_{0i}}, \texttt{Verf}_{x_{0i}})$ and send $g^{k_{0i}}$ and $\texttt{Verf}_{x_{0i}}$ to the CAs via a secure channel to upload them on the blockchain and keep the private key to itself. There will be multiple number of CAs to avoid single point of failure and also to decentralize the system. The CAs will upload the public keys assigned with the IDs along with the public key certificate. The devices can read the public keys of all the other devices from the blockchain itself. The CAs will use `ECDSA` to generate cerificates corresponding to the public keys. It will look like $m||S_{sk}(m)$ where $m = ID||Public\ Key$. The CAs will assign unique IDs to the devices according to the requirement of that particular IoT environment and afterwards the devices themselves can construct the IDs for their parent node and all the above intermediate nodes that they are the child nodes of, independently, and the CAs can verify whether the device belongs to that group, i.e., it has access to generate the public keys for that abstract node. These IDs are then mapped to the leaf nodes of a binary tree based on a required structure, determining sibling relationships and node placement.

II. Setup:

- Each device x first fetches the public key of $s(x)$ from the blockchain with the desired ID by calling the smart contract. 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.
- Each device x computes the secret key for $p(x)$ 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

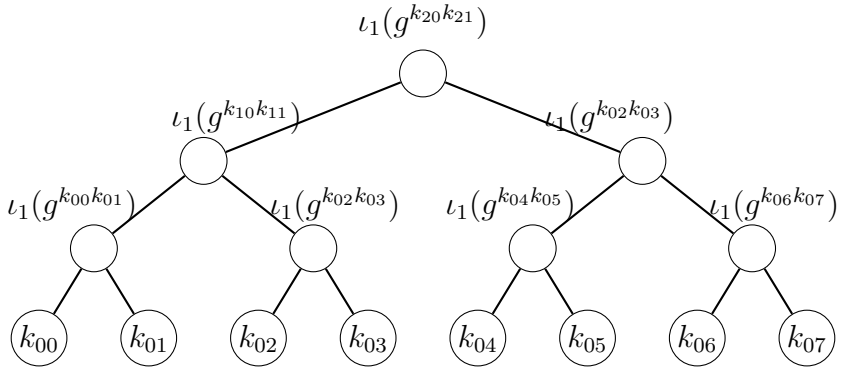


Figure 4.2: Recursive key derivation in a binary tree structure

x_{mn} as shown in Figure 4.1. 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.

- The secret key of the parent node $x_{1(\lfloor n/2 \rfloor)}$ is computed as:

$$sk_{\text{parent}} = \iota_1\left((g^{k_2})^{k_1}\right) = \iota_1\left(g^{k_1 k_2}\right),$$

- The corresponding public key of the parent node becomes:

$$pk_{\text{parent}} = g^{sk_{\text{parent}}} = g^{\iota_1(g^{k_1 k_2})}.$$

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

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

- Then the computed public key will be sent to the CAs and they will check whether the device is a leaf node of that subtree, rooted at that abstract node. The public key will be uploaded to the blockchain with the identity tag corresponding to the $x_{1(\lfloor n/2 \rfloor)}$ node.
- Again, the device will retrieve the public key of the sibling node of its parent node from the blockchain and compute the secret key and public key of the node $x_{2(\lfloor \lfloor n/2 \rfloor /2 \rfloor)}$ and assign the appropriate identity with it and send it to the CAs to upload it to the block chain.
- The process continues recursively to compute the secret key of the root node of the binary tree, as illustrated in Figure 4.2.

III. 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.
Achieving PFS is straightforward in one-to-one communication via ephemeral Diffie–Hellman key exchange, but becomes challenging in one-to-many communication.

In our protocol, we therefore provide PFS for one-to-one communication and *partial forward secrecy* for one-to-many communication. We could not achieve perfect forward secrecy in multicast communication.

- **Without PFS:**
  If PFS is not required in our IoT environment, then we can use the following session key establishment method. Let A be a sender device with a long-term secret key α (so its public key is g^α), and let B be a receiver, either a single device, a subset of the group, or the entire group with the long-term secret key β (public key g^β). We derive a session key at round $i \in \mathbb{Z}_{\geq 0}$ by

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

  or for broadcasting,

  $$K^i_{\text{session}} \;=\; \text{KDF}\Big(tk \,\|\, i\Big).$$

  If an adversary compromises A or B and learns α or β, then they can immediately recompute $g^{\alpha\beta}$ and hence recover all the keys of the previous session. However, learning a single session key K^i_{session} at time i does not allow the adversary to recover α, β, or any previous session keys, since inverting the KDF is assumed computationally hard.
- **PFS in one-to-one communication:**
  In one-to-one communication, we achieve perfect forward secrecy by combining the long-term key of each party with fresh ephemeral exponents. Concretely:
  - 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

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

  Because each session uses new ephemeral keys, even if an adversary later compromises long-term secret of one party (say a), they cannot reconstruct any past session keys, those depend on one-time secrets x and y, which are securely erased after use. This guarantees perfect forward secrecy in one-to-one communication.
- **Partial FS:**
  In one-to-many communication, we achieved partial forward secrecy by combining each party's long-term key with fresh ephemeral exponents. Concretely:
  - Let a and b be the long-term secret keys of parties A and B where A is a single device and B is a subset of the entire group or the entire group, 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.

– Then both parties derive the shared session key as

$$K_{\text{session}} = H\left(g^{ab} \,\|\, g^{xb}\right).$$

Because each session uses a new exponent x , even if an adversary later compromises the sender's long-term secret key a, they cannot reconstruct any past session keys, those depend on the one-time secret x which are securely erased after use. On the other hand, if the adversary compromises the receiver's long-term private key, then he/she can reconstruct all the past session keys. That is why we are calling this partial forward secrecy.

- **Why no PFS in one-to-many communication:** In a one-to-many communication, we cannot achieve perfect forward secrecy because it would require an ephemeral–ephemeral key exchange between the sender A and one from the entire group B which is same as generating a single *group ephemeral key* in each session independent of the long-term secrets. which will increase the computational and communication cost. Hence, we settle for *partial* forward secrecy in multicasting.

IV. Key Update

Step 1: Update Trigger: A periodic or event-driven key update is initiated for leaf node x_{06} in Figure 4.3.

Step 2: Leaf Key Regeneration: Node x_{06} generates a fresh private–public key pair $\left(k'_{06},\, g^{k'_{06}}\right)$.

Step 3: Path Rekeying: Node x_{06} recomputes all intermediate secret keys and corresponding public keys along the path(x_{06}) and are sent to the CAs, which upload them to the blockchain.

Step 4: Distribution of New Keys: Node x_{06} encrypts each new secret key under the public key of the nodes in copath(x_{06}) and multicasts these secret keys to the existing devices.

Step 5: Verification: All recipients retrieve the updated public keys from the blockchain, decrypt their shares, and verify the consistency of the rekey operation.

V. Revocation

Step 1: Revocation Trigger: A device is revoked from the system, here x_{37} as shown in Figure 4.3. (e.g., due to compromise).

Step 2: Certificate Revocation: The CAs revoke the certificate of the revoked device x_{07} and update the list on the blockchain.

Step 3: Sibling Key Update: The sibling node x_{06} of the revoked leaf node performs the Key Update method described above.

VI. Addition

Device addition procedure is similar to the revocation method.

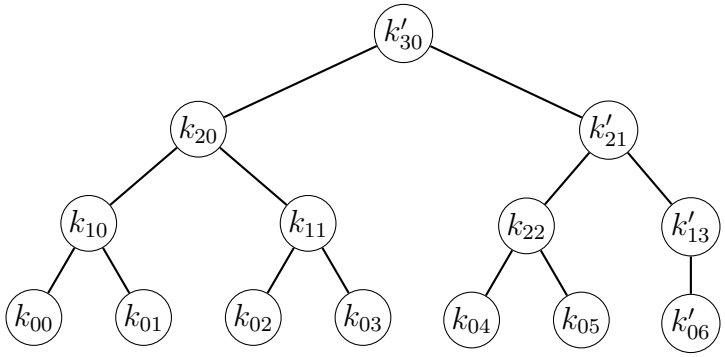


Figure 4.3: Updated Tree after Revocation of x_{37} and Re-Keying by x_{36}

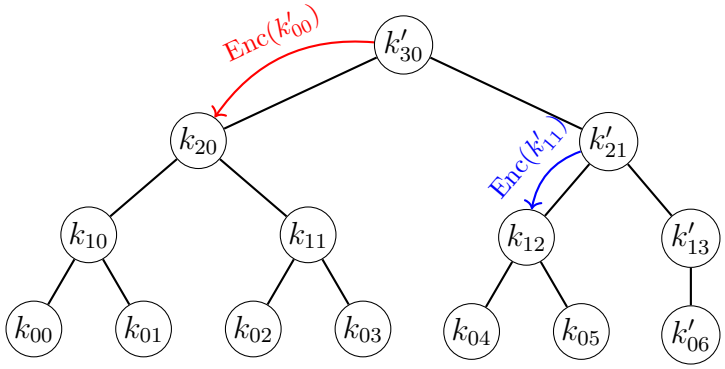


Figure 4.4: Updated Tree after Revocation of x_{07} and Re-Keying by x_{06}

4.1.4 Computational Cost:

Let n be the total number of devices and $h = \lceil \log_2 n \rceil$ be the height of the binary tree. For ease of computation, we assume $n = 2^m$ for some non-negative integer m, so that the tree is perfectly balanced.

Cost per Device

Each device in a leaf performs the following operations:

1. **Generating its own Public Key:**
   The device does one exponentiation to compute:
   $$pk_{\text{leaf}} = g^{sk_{\text{self}}}$$
2. **For each of the h levels of the tree (to compute keys up to the root):**
   - One exponentiation using the sibling's public key:
     $$a = (pk_{\text{sibling}})^{sk_{\text{self}}}$$
   - One hash computation to derive the parent secret key:
     $$sk_{\text{parent}} = \iota_1(a)$$
   - One exponentiation to compute the parent public key:
     $$pk_{\text{parent}} = g^{sk_{\text{parent}}}$$

Total Operations per Device:

$$\begin{aligned}\text{Integer Function evaluations} &= h = \lceil \log_2 n \rceil, \\ \text{Exponentiations} &= 1 \text{ (for leaf public key)} \ + \ 2h \text{ (per level)} \\ &= 1 + 2\lceil \log_2 n \rceil.\end{aligned}$$

Let us define:

T_{root}: the time to evaluate the secret key of the root node.

At each level (except for the 0th level), the following operations are performed:

- One integer function evaluation,
- Two exponentiation.

Therefore, for each level of the tree (except for the 0^{th} level), the total time required to calculate the secret key of a node is:

$$T_{\iota_1} + 2T_e$$

For the 0th level, there is only one exponentiation for every leaf, so it only requires T_e time to generate the public key of that leaf.

However, since all devices at a given level operate in parallel, the time complexity is determined by the number of sequential levels (i.e., the height of the tree h).

Hence, the total time required to compute the secret key and public key of the root node is:

$$T_{\text{root}} = h \cdot (T_{\iota_1} + 2T_e) + T_e = hT_{\iota_1} + 2hT_e + T_e = hT_{\iota_1} + (2h + 1)T_e$$

4.1.5 Space Requirement:

In this protocol, each device stores secret keys of the nodes in $path(x)$ of the binary tree. For a system with n devices, the height of the tree is $\log_2 n$, and therefore each device needs to store $\log_2 n$ secret keys. They will fetch the public keys of all the other nodes from the blockchain when necessary.

Assuming each secret key is 128 bits, the total storage required per device is:

$$128 \times \log_2 n \quad \text{bits} \quad = \quad \frac{128 \times \log_2 n}{1000} \quad \text{Kb} = 0.128 \log_2 n \text{ Kb}\,.$$

Where many Micro Controller Units used in IoT (e.g., Wio Terminal) offer around 192KB of SRAM.

4.1.6 Remarks:

- We tried to generalize our protocol following the approach of Katz et al. [20].
  Let $r_1, \ldots, r_n \xleftarrow{R} \mathbb{Z}_q$;

  Compute:

$$z_1 = g^{r_1}, \quad z_2 = g^{r_2}, \quad \ldots, \quad z_n = g^{r_n};$$

$$\Gamma_{1,2} = g^{r_1 r_2}, \quad \Gamma_{2,3} = g^{r_2 r_3}, \quad \ldots, \quad \Gamma_{n-1,n} = g^{r_{n-1} r_n}, \quad \Gamma_{n,1} = g^{r_n r_1};$$

$$X_1 = \frac{\Gamma_{1,2}}{\Gamma_{n,1}}, \quad X_2 = \frac{\Gamma_{2,3}}{\Gamma_{1,2}}, \quad \ldots, \quad X_n = \frac{\Gamma_{n,1}}{\Gamma_{n-1,n}};$$

  The secret key:

$$\mathsf{sk} = (\Gamma_{n,1})^n \cdot (X_1)^{n-1} \cdots X_{n-1} = g^{r_1 r_2 + r_2 r_3 + \cdots + r_n r_1};$$

  This method is not scalable for resource-constrained devices, as the number of operations for each device grows linearly with the number of devices. Moreover, broadcasting a large number of public values can saturate the limited bandwidth typical of IoT networks.
- To add an extra layer of security towards PFS, in a closed environment, such as a smart home, where PFS is necessary, we can incorporate a context-based nonce collected from the environment, similar to the approach used in IOTCUPID [14]. This nonce can then be utilized in the generation of session keys.
  If we can ensure that the adversary has no physical access to the interior of the home, then even after compromising the secret key of any device, the adversary can only obtain the past session keys they were already aware of the past environment nonces. Although this does not provide PFS, it offers a security enhancement towards achieving PFS.
  We can also treat the secret keys of the root and intermediate nodes of the tree as passwords, and by using partitioned GPAKE, we can derive a new shared group session key, which is suitable for deployment in small IoT environments, such as smart homes.
- We provided a generalization of our protocol for nested IoT environments, i.e., think smart buildings or even an entire smart city, by organizing devices into a hierarchical structure using a binary tree. Using the same blockchain-based PKI and our proposed protocol, devices sharing a physical location (e.g., the same room or floor) derive a common secret. This kind of structure will help us to maintain efficient key management across all layers of the system. By assigning the proper IDs to each device, we can realize this structure. We can also use this idea in multi-domain IoT environments such as industrial IoT systems.

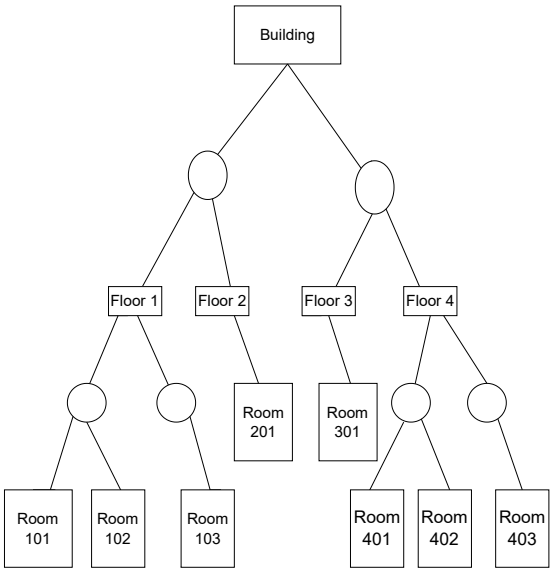


Figure 4.5: Illustration of generalization of Protocol 1 in a nested environment

4.2 Protocol 2

The motivation behind this protocol was initially to reduce the height of the tree, thereby minimizing the overall computational cost per device. However, in practice, the use of bilinear pairings introduces significant overhead (discussed in [I.]), especially in resource-constrained IoT devices, where such operations are computationally expensive and time consuming. However, this protocol presents an alternative approach to address the same group key establishment problem.

This construction is structurally similar to the first protocol based on a binary tree, with the primary difference being the use of a ternary tree for organizing devices. The secret key of the root node is derived using a *tripartite Diffie–Hellman* key exchange [19], using the values contributed by its three child nodes. We highlight the modifications made to the key computation procedure compared to the binary tree approach.

All other mechanisms, including initialization, device addition, and secure revocation, follow the same procedures as in the binary tree protocol.

4.2.1 Threat Model

Similar threat model as the previous one.

4.2.2 Identity Assignment

Similar to the previous one. So, we have the following method.

$$\text{ParentID}(x_{mn}) = (m+1) \,||\, \left\lfloor \frac{n}{3} \right\rfloor,$$

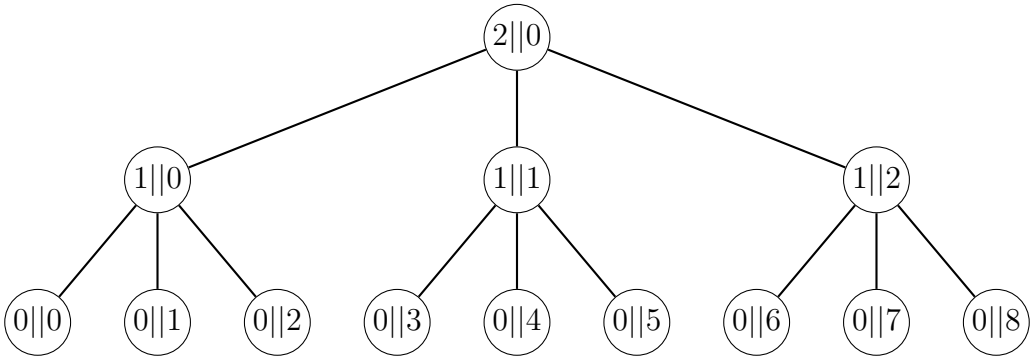


Figure 4.6: Identities of the devices in the ternary tree for simplicity

4.2.3 Protocol:

I. Initialization

Same as in Protocol 1.

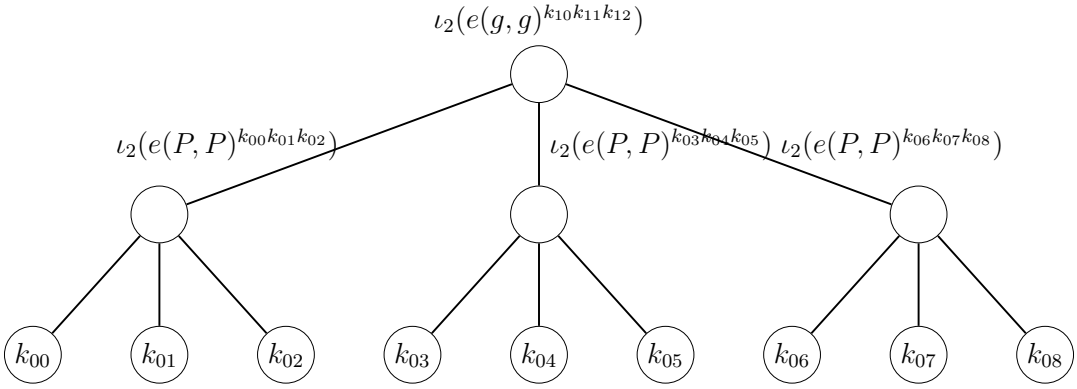


Figure 4.7: Recursive key derivation in a ternary tree using tripartite Diffie-Hellman and hash functions.

II. Setup:

- Each device first fetches the public key of its sibling nodes from the block chain with the desired ID by calling the smart contract. If it has ID $0||n$ where n is of the form $3k$ for some nonnegative integer k then it will ask for the public key of the device with identity $0||(n+1)$ and $0||(n+2)$. If n is of the form $3k+1$ for some nonnegative integer k, then it will ask for the public key of the device with the identities $0||(n-1)$ and $0||(n+1)$. If n is of the form $3k+2$ for some nonnegative integer k, then it will ask for the public key of the device with the identities $0||(n-1)$ and $0||(n-2)$.
- Each device computes the secret key for its parent (an abstract internal node) using the public key of its sibling nodes in the tree and its secret key. Let z be the secret key of x_{0n}, and let xP_1, yP_1 be the public keys of its sibling nodes.
- The secret key of the parent node $x_{1(\lfloor n/3 \rfloor)}$ is computed as:

$$sk_{\text{parent}} = \iota_2\left(e(xP_1, yP_1)^z\right) = \iota_2\left(e(P_1, P_1)^{xyz}\right),$$

- The corresponding public key of the parent node becomes:

$$pk_{\text{parent}} = sk_{\text{parent}} P_1 = \iota_2 \left(e(P_1, P_1)^{xyz} \right) P_1$$

- If a device has exactly one sibling node, it derives the public key for the corresponding abstract parent node using the *Diffie-Hellman key exchange* protocol. Specifically, the key is computed as in the previous protocol:

$$sk_{\text{parent}} = i_2 \left(y(xP_1) \right) = i_2 \left(xyP_1 \right)$$

  where x and y are the private keys of the two sibling nodes.
- Again, if a device x has no sibling node, then it selects $k \xleftarrow{R} \mathbb{Z}_q$

$$sk_x = k = sk_{p(x)}$$

- This public key is uploaded to the blockchain with the identity tag corresponding to $x_{1(\lfloor n/3 \rfloor)}$ node.
- Again, the device will retrieve the public key of the sibling nodes of its parent node from the blockchain and compute the public key of the node $x_{2(\lfloor \lfloor n/3 \rfloor /3 \rfloor)}$ and assign the proper identity with it and send it to the CAs to upload it to the block chain.
- The process continues recursively to compute the secret key of the root node of the ternary tree, as illustrated in Figure 4.1.

III. **Session Key Establishment:**

This follows the same process as setting the session key, as described in III.

4.2.4 Computational Cost:

Let n be the total number of devices, and let $h' = \lceil \log_3 n \rceil$ denote the height of the ternary tree. For ease of computation, we assume $n = 3^k$ for some integer k, so that the tree is perfectly balanced.

Number of Computations per Device:

Each leaf device performs the following operations:

1. **Generating its own Public Key:**
   The device performs **one exponentiation** to compute:

$$pk_{\text{leaf}} = sk_{\text{self}} P_1$$

2. **For each of the h levels of the tree:**
   At every level, while computing the keys for parent nodes, the device performs:
   - **One bilinear pairing evaluation:**

$$a = e(pk_{\text{sibling1}}, pk_{\text{sibling2}})$$

- **One exponentiation:**
$$\beta = a^{sk_{\text{self}}}$$
- **One hash computation:**
$$sk_{\text{parent}} = \iota_2(\beta)$$
- **One exponentiation:**
$$pk_{\text{parent}} = sk_{\text{parent}} P_1$$

Total number of operations:

$$\begin{aligned} \text{Hash evaluations} &= h' = \lceil \log_3 n \rceil \,, \\ \text{Exponentiations} &= 1 + 2h' = 1 + 2\lceil \log_3 n \rceil \,. \\ \text{pairing evaluations} &= h' = \lceil \log_3 n \rceil \,, \end{aligned}$$

We now analyze the total time required to compute the secret key at the root of the tree.

Let us define:

T'_{root}: The time to evaluate the secret key of the root node.

At each level (except for the 0^{th} level), the following operations are performed:

- One hash function evaluation,
- Two exponentiation.

Therefore, for each level of the tree (except for the 0^{th} level), the total time required to calculate the secret key of a node is:

$$T_{\iota_2} + 2T_e + T'_p$$

For the 0^{th} level, there is only one exponentiation for each leaf, so it only takes T_e time to generate the public key of that leaf.

However, since all devices at a given level operate in parallel, the time complexity is determined by the number of sequential levels (i.e., the height of the tree h).

Hence, the total time required to compute the secret key and public key of the root node is:

$$T'_{\text{root}} = h \cdot (T_{\iota_2} + 2T_e + T_p) + T_e = hT_{\iota_2} + 2hT_e + hT_p + T_e = hT_{\iota_2} + (2h+1)T_e + hT_p$$

4.2.5 Space Requirement:

For a system with n devices, the height of the tree is $\log_3 n$, and therefore each device needs to store $\log_3 n$ secret keys.

Assuming each secret key is 128 bits, the total storage required per device is:

$$128 \times \log_3 n \quad \text{bits} \quad = \quad \frac{128 \times \log_3 n}{1000} \quad \text{Kb} = 0.128 \log_3 n \ \text{Kb} \,.$$

4.2.6 Remarks:

We have tried to generalize this idea using the following method.

- **Multilinear Maps** Existing multilinear maps such as GGH13 [17] and CLT13 [11] require intensive operations (zero testing, large integer arithmetic) of the order of hundreds of milliseconds to seconds per evaluation on moderately powerful hardware. This makes them impractical for resource-constrained IoT devices. We also have some statistical attacks on existing multilinear maps [8, 12]. Otherwise, we could develop asymmetric n-ary key agreement trees leveraging multilinear maps. In such a structure, all devices within the same domain could be placed under a common parent node.

Chapter 5

Security Analysis

In this chapter, we present a comprehensive security analysis of our protocol, evaluating its resilience against both passive adversaries and a wide range of active attacks, including MITM attacks, replay attacks, known-key attacks, and others.

5.1 Security Against Passive Adversary

Now, we will show that our protocol is secure against a passive adversary.

Let $\mathbb{G}$ be a cyclic group of prime order q with generator g. Let $H : \mathbb{G} \to \mathbb{Z}_q$ be a Hash function. Define the following security game:

❑ **XY Game:** An adversary $\mathcal{A}$ is given a tuple

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

for random $x, y \xleftarrow{\$} \mathbb{Z}_q^*$, and must output a guess $T' \in \mathbb{Z}_q$ for the value $H(g^{xy})$.

Define its *XY advantage* as:

$$\mathrm{Adv}_{\mathrm{XY}}(\mathcal{A}) \ = \ \mathbb{P}\Big[\mathcal{A}(g, g^x, g^y, g^{H(g^{xy})}) = H(g^{xy})\Big].$$

Lemma 5.1. *If the HDDH assumption holds in $\mathbb{G}$, then no PPT adversary $\mathcal{A}$ can solve the XY problem with non-negligible advantage. Equivalently, for all PPT $\mathcal{A}$ there exists a negligible function $\mathsf{negl}(\cdot)$ such that*

$$\mathrm{Adv}_{\mathrm{XY}}(\mathcal{A}) \ \leq \ \mathsf{negl}(\lambda).$$

Proof. Suppose that there exists a PPT adversary $\mathcal{A}$ with

$$\mathrm{Adv}_{\mathrm{XY}}(\mathcal{A}) \ \geq \ \tfrac{1}{q} \ + \ negl(\lambda)$$

We build a distinguisher $\mathcal{D}$ that breaks the HDDH assumption with advantage at least ϵ.

1. $\mathcal{D}$ receives a challenge tuple (g, g^x, g^y, T) from the HDDH challenger.

2. $\mathcal{D}$ computes g^T in the group $\mathbb{G}$, and forwards (g, g^x, g^y, g^T) to $\mathcal{A}$.
3. $\mathcal{A}$ returns a candidate $T' \in \mathbb{Z}_q$.
4. If $T' = T$, $\mathcal{D}$ outputs 1 ; otherwise it outputs 0 .

Now, If $T = H(g^{xy})$, then the view of $\mathcal{A}$ is identically the XYZ game, so

$$\mathbb{P}\Big[\mathcal{D} \text{ outputs } 1 \mid T = H(g^{xy})\Big] = \mathbb{P}\Big[\mathcal{A} \text{ outputs } H(g^{xy})\Big] = \mathrm{Adv}_{\mathrm{XYZ}}(\mathcal{A}).$$

If $T = r \xleftarrow{R} \mathbb{Z}_q$, then g^T is independent of x, y, so any guess T' succeeds with probability exactly $1/q$. Hence

$$\mathbb{P}\Big[\mathcal{D} \text{ outputs } 1 \mid T = r\Big] = \frac{1}{q}.$$

Therefore the HDDH advantage of $\mathcal{D}$ is

$$\begin{aligned}\mathrm{Adv}_{\mathrm{HDDH}}(\mathcal{D}) &= \Big|\mathbb{P}[\mathcal{D} = 1 \mid T = H(g^{xy})] - \mathbb{P}[\mathcal{D} = 1 \mid T = r]\Big| \\ &= \Big|\mathrm{Adv}_{\mathrm{XYZ}}(\mathcal{A}) \;-\; \tfrac{1}{q}\Big| \;\geq\; \Big(\tfrac{1}{q} + negl(\lambda)\Big) - \tfrac{1}{q} \;=\; negl(\lambda),\end{aligned}$$

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

which is non-negligible. This contradicts the HDDH assumption. □

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 $n \leq 2^h$.

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

Define:

- $\mathsf{view}(h, K, B) := \{g^{k_{ij}} \mid \forall i, j\}$ — public information observed by a passive adversary from the blockchain.
- $\mathrm{gk} = x_{h0} = \iota_1\left(g^{x_{(h-1)0}x_{(h-1)1}}\right)$ — secret key of the root node.

Theorem 5.1. *Assuming the HDDH problem is computationally hard in $\mathbb{G}$, we assert that:*

No PPT adversary can compute gk given $\mathsf{view}(h, K, B)$.

Proof. The structure of the proof is similar to that of Barua et al. [5].

Base Case: $h = 1$

By Lemma 5.1, we see that the problem **XY** is harder than HDDH problem 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 $\mathsf{view}(h, K, B)$.

□

5.2 Security Against Active Adversary

Now, we will show how our protocol also defends against active adversaries.

- **Man-in-the-Middle (MITM) Attack:** In our protocol, each device transmits its public key to the CAs using a secure channel. When a device later retrieves the public key of an internal node associated with the required identity from the blockchain, it first verifies the CA's digital signature. Only after successful verification of the signature, the device independently computes the public key of the parent node corresponding to that internal node.
  Since every public key is signed by trusted CAs and is immutably published on the blockchain, any attempt by an adversary to intercept, modify, or inject misinformation onto the blockchain can be immediately detected during the signature verification process. This end-to-end integrity mechanism ensures that MITM attacks cannot compromise our tree-based key derivation process.
  In the partial PFS scenario, the sender generates an ephemeral key and transmits it along with a digital signature. The receivers can then easily verify the authenticity of the ephemeral key using the sender's verification key.
- **Replay Attack & Key Freshness:** In our session-key establishment protocol, each new session derives a fresh, random key through a KDF that incorporates a unique session nonce, such as a timestamp or a session counter, in scenarios without PFS.
  In the case of partial PFS, the sender generates a random ephemeral key and transmits it to the receivers. Then both parties compute the session key using the shared secret key and the ephemeral key.

Since every session key is unpredictable and never used before, any attempt by an adversary to replay old messages will fail, the ciphertext will decrypt incorrectly under the new session key, or be rejected outright. Thus, the protocol will be resistant to replay attacks and will also guarantee key freshness.

- **Key Confirmation:** In a scenario without PFS , each device independently generates session keys using the same mechanism. Whenever one device increments the session counter, it informs the other devices using the previous session keys, enabling them to increment their session counters as well. As a result, all devices compute the exact same session key for each session.
  In a partial PFS scenario, the receivers verify the ephemeral key using the sender's verification key. Subsequently, they compute the session key based on the sender's ephemeral key and the long-term public keys of both the sender and the receivers. Thus, all parties derive the same session key.
- **Key Control:** In our protocol, each device generates its own long-term secret key and uploads the corresponding public key to the blockchain through CAs. Using the public key of a neighboring node, which ultimately depends on the neighbor's secret key and the device's own secret key, the devices collaboratively derive the secret key of their parent node. Consequently, no entity can predict or manipulate the final secret key gr or the secret key of any other internal node.
  The secret key of every internal node is inherently dependent on the secret keys of all its child nodes. Thus, the secret keys of the internal nodes, including the root node, are the result of a joint contribution of their respective child nodes.
  In scenarios without PFS, session keys are derived by concatenating the session counter with the shared secret. This ensures that no device can manipulate the session keys. Similarly, in the partial PFS scenario, session keys depend on both the long-term public keys of the receivers and the ephemeral keys generated by the senders.
- **Perfect Forward Secrecy:** We have discussed forward secrecy in the Session Key Establishment section (Section III.). Our protocol does not achieve PFS in multicasting. However, we were able to achieve partial forward secrecy, wherein the compromise of the long-term secret key of the sender does compromise the past session keys. In contrast, if the secret key of the receiver is compromised, an adversary can recover all past session keys.
  On the other hand, our protocol achieves PFS in unicast communication. This ensures that all past one-to-one communications remain secure, even if the long-term secrets of both the sender and the recipient are compromised.
- **Post Compromise Security:** Post-compromise security is also referred to as self-healing. If the secret key of a device is compromised, the adversary can compute all secret keys along the $path(x)$, where x denotes the compromised device. To mitigate this, all secret keys along $path(x)$ must be updated.
  In this scenario, the device generates a new secret key for itself, recomputes all secret keys along $path(x)$, and sends the updated secret keys to the CAs. The device also informs the other existing devices accordingly.
  In one-to-one communication, the use of ephemeral secrets further enhances security by protecting future sessions, even if the long-term key has been compromised.

- **Resistance to Known-key Attack:**
  We derive each session key by:

$$K^{i}_{\text{session}} \;=\; \text{KDF}(\mathsf{gk}||\mathsf{i}),$$

  where KDF or H is assumed to be preimage resistant. Hence, even if an adversary learns all the previous session keys $K^{1}_{session}, K^{2}_{session}, \ldots, K^{i-1}_{session}$, the preimage resistance of KDF or H prevents the recovery of the corresponding inputs. Consequently, the adversary cannot compute the future session-key $K^{i}_{session}$.
- **Mutual Authentication & Unilateral Authentication:** During session-key establishment using ephemeral key exchange in multicast or broadcast scenarios, the sender signs its ephemeral public key with its private signature key. Each receiver retrieves the corresponding verification key from the blockchain and verifies the signature. In one-to-one communications, both devices sign the ephemeral public key with their respective private signature keys, thereby achieving mutual authentication.
- **External Anonymity:** Currently, Our protocol does not achieve external anonymity. To address this limitation, Pseudo Identities (PIDs) can be used by Hashing actual identities along with nonces to anonymize the actual identities of the devices, as discussed by Liu et al. [24]

$$PID = H(ID||\xi)$$

  , where ξ is nonce.

Chapter 6

Evaluation

The security and efficiency of any cryptographic protocol must be rigorously evaluated to justify its practical applicability, especially in resource-constrained environments. This chapter presents a complete assessment of the proposed protocol, focusing on its computational cost and communication efficiency.

Table 6.1: Typical Cryptographic Operation Timings on Constrained IoT Devices

	Hash Evaluation	**Scalar Multiplication**	**Pairing Evaluation**
Typical Time	<1 ms	10–30 ms	100 ms–several seconds

Notes:

- SHA-256 on 32-bit microcontrollers often completes within **<1 ms** for inputs around 100–200 bytes using standard libraries (e.g., mbedTLS) [*SHA256*].
- Elliptic curve scalar multiplications (ECDH) on devices such as Cortex-M4 typically take about **10–30 ms**, with average reports near 15 ms for 256-bit curves [ECDH].
- Bilinear pairings on low-power hardware range from **hundreds of milliseconds to multiple seconds**, depending on parameters and optimizations [**?**].

† Pairing times vary significantly depending on the curve choice and hardware, some require up to several seconds on early IoT boards; optimized implementations on faster Cortex-M33 or M4 devices can be under 100ms but are still much slower than ECC operations.

I. Comparison of Protocol 1 and Protocol 2:

We have two different expressions representing the total time required to compute the root secret and the corresponding public key for the binary tree and the ternary tree, respectively.
:

$$T_{\text{root}} = h\,T_H + (2h+1)\,T_e,$$
$$T'_{\text{root}} = h'\,T_H + (2h'+1)\,T_e + h'\,T_p,$$

where

$$h = \log_2 n, \qquad h' = \log_3 n,$$

and clearly $h' < h$.
We require $T'_{\text{root}} \le T_{\text{root}}$. Equivalently,

$$\begin{aligned} T'_{\text{root}} - T_{\text{root}} \le 0 &\iff (h'-h)\,T_H + 2(h'-h)\,T_e + h'\,T_p \le 0 \\ &\iff (h'-h)\,(T_H + 2T_e) + h'\,T_p \le 0 \\ &\iff h'\,T_p \;\le\; (h-h')\,(T_H + 2T_e) \\ &\iff T_p \;\le\; \left(\tfrac{h}{h'} - 1\right)(T_H + 2T_e). \end{aligned}$$

Since $h = \log_2 n$ and $h' = \log_3 n$, we have

$$\frac{h}{h'} \;=\; \frac{\log_2 n}{\log_3 n} \;=\; \frac{\ln n/\ln 2}{\ln n/\ln 3} \;=\; \frac{\ln 3}{\ln 2} \;=\; \log_2 3 \;>\; 1.$$

Thus

$$\frac{h}{h'} - 1 = \log_2 3 \;-\; 1 < \log_2 3 \quad\Longrightarrow\quad \left(\tfrac{h}{h'} - 1\right)(T_H + 2T_e) < (\log_2 3)\,(T_H + 2T_e) < (T_H + 2T_e).$$

Hence, T_p must satisfy

$$T_p \;<\; (T_H + 2T_e)$$

to ensure $T'_{\text{root}} \le T_{\text{root}}$.
Now, taking conservative (minimum) realistic values

$$T_H = 1 \text{ ms (for ease of computation)}, \quad T_e = 10 \text{ ms}, \quad T_p = 100 \text{ ms},$$

the inequality becomes

$$T_p \;<\; (T_H + 2T_e) \implies T_p \;<\; (1 + 2 * 10) = 21,$$

which is not true because we have $T_p = 100$ ms. Hence, there is no integer $n > 1$ that satisfies the inequality.
Therefore, under realistic timing parameters, Protocol 2 (with pairings) is always slower than Protocol 1.

II. Scalability:

Moreover, from the logarithm table, it becomes evident that the proposed protocol is highly efficient for a large-scale IoT environment. For example, in a network comprising 10^6 IoT devices, each device performs merely 20 hash computations and 41 modular exponentiations.

Table 6.2: Logarithm Table with Base 2

x	$\log_2(x)$	x	$\log_2(x)$
1	0.00000	100	6.64386
2	1.00000	200	7.64386
3	1.58496	500	8.96578
4	2.00000	1000	9.96578
5	2.32193	2000	10.96578
6	2.58496	5000	12.28771
7	2.80735	10000	13.28771
8	3.00000	20000	14.28771
9	3.16993	50000	15.60964
10	3.32193	100000	16.60964
50	5.64385	1000000	19.93157

III. Comparison with other works:

We have explained the existing protocols in Literature Review chapter [2]. Now, we will compare those protocols with our proposed protocol. The table below summarizes the comparison between several existing works and our proposed protocol.

Protocol	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8
JEDI [21]	✓	✓	✗	✓	✙	✗	✗	✓
IoTCUPID [14]	✗	✗	✓	✓	-	-	✓	✗
Perceptio [3]	✗	✗	✗	✓	-	✓	✓	✗
T-HIBE [13]	✓	✗	✓	✓	✗	✓	✓	✗
Symmetric Key [26]	✓	✗	✓	✗	✗	✓	✓	✗
VSS [25]	✗	✗	✗	✗	-	✓	✓	✗
Proposed Work (Protocol 1)	✓	✓	✓	✓	◐	✓	✓	✙

Table 6.3: Comparative Analysis of Protocols Across Key Security Properties

Property Assignments:

P_1 = Scalability,
P_2 = Efficient Immediate Revocation,
P_3 = Avoid single TTP,
P_4 = Decentralized,
P_5 = Perfect Forward Secrecy (PFS),
P_6 = Post Compromise Security,
P_7 = Efficiency for Constrained Devices,
P_8 = External Anonymity.

Symbols for Protocol Properties

- ✓ Protocol **fully supports** the property
- ✗ Protocol **does not support** the property
- ◐ Protocol **partially supports** the property
- ✙ Protocol **can be extended** to support the property

† For JEDI, the size of the ciphertext increases with the number of revoked devices, whereas in T-HIBE, the size of the ciphertext is always large regardless of revocation.

Chapter 7

Future Work & Conclusion

7.1 Future Work

Although the proposed protocol addresses decentralized group key establishment and secure communication in resource-constrained IoT environments, several aspects have not yet been improved.

Although our proposed protocol provides partial forward secrecy, in certain scenarios, such as smart homes, we need perfect forward secrecy to secure the past communications among IoT devices. To address this limitation, the protocol can be modified to incorporate mechanisms that guarantee PFS.

An important direction is to generalize our protocol to an *asymmetric tree structure* suitable for nested environments (e.g., Room → Floor → Building) in smart buildings and smart cities. In such hierarchies, devices with frequent interactions are grouped as child nodes under a common parent node.We have briefly discussed this generalization in the Remarks section. However, if there are so many levels in that hierarchy of that IoT environment and a huge number of IoT devices, then the height of the tree will grow accordingly, which in turn increases the computational cost.

An interesting extension would be the incorporation of external anonymity using pseudo identities, which enables devices to remain anonymous in open environments while still participating securely in group communication.

In addition, although blockchain-based PKI enhances trust and verifiability, it introduces additional infrastructure costs. Exploring other PKI models suitable for small-scale IoT environments such as smart homes can make the protocol more practical in such scenarios.

Another direction for future work could be the complete removal of the CAs, which allows IoT devices to directly upload their own public keys along with identities to the blockchain. By carefully designing the group formation process, we can ensure that only legitimate devices can participate or can not disrupt the tree structure, preventing adversaries from falsely claiming group membership or injecting fake nodes. This approach would significantly improve scalability and eliminate dependency on trusted third parties.

Finally, although the proposed protocol is supported by rigorous mathematical proofs and formal security models, practical validation through real-world deployment and testing will also be pursued. This will help to verify that the protocol maintains robustness, scalability, and efficiency in practice.

7.2 Conclusion

Securing communication between IoT devices without relying on a trusted central third party is essential, but traditional protocols often fall short, either lacking scalability, efficiency in the device revocation method, or efficiency for resource-constrained environments. Centralized models are vulnerable to a single point of failure, making decentralized solutions crucial.

In this work, we have proposed two secure decentralized group key establishment protocols. By organizing devices in tree-like structures and leveraging hybrid key establishment mechanism that uses public-key cryptography to securely generate session keys, which will be treated as symmetric keys. Our approach ensures secure and scalable communication while minimizing computational overhead.

The first protocol uses a binary tree and Diffie-Hellman key exchange to derive group keys in a bottom-up manner. The second protocol extends this idea using a ternary tree and tripartite Diffie-Hellman key exchange to reduce tree height and improve efficiency, although it introduces higher computational cost due to bilinear pairings. We discussed an idea to generalize the proposed protocol.

To further enhance security and trust, we introduced a blockchain-based public key infrastructure (PKI), providing tamper-proof certificate management and eliminating reliance on a single certificate authority. Additionally, the use of recursive key computation and localized updates ensures that the protocol can efficiently handle dynamic events such as device addition or revocation without the need for reinitialization of the entire system.

Our protocols offer strong security guarantees, including authentication, post-compromise security, and partial forward secrecy; they also maintain practicality for real-world IoT systems by adopting a hybrid mechanism that uses public-key cryptography to generate lightweight symmetric session keys.

Although our proposed protocol satisfies a wide range of security properties, including resistance to replay and MITM attacks, and session key confidentiality, it still lacks certain privacy measures that are essential in privacy-sensitive applications. In particular, the proposed protocol currently does not support *external anonymity* and also *PFS*. These limitations open up meaningful directions for future enhancement.

In general, this work contributes to a scalable, decentralized, and efficient key management framework that addresses the unique challenges of secure communication in large-scale, heterogeneous, and resource-constrained IoT environments.

Bibliography

[1] Abdalla, M., Bellare, M., Rogaway, P.: DHAES: An encryption scheme based on the Diffie-Hellman problem. Cryptology ePrint Archive, Paper 1999/007 (1999), https://eprint.iacr.org/1999/007

[2] Abdalla, M., et al.: Generalized Key Delegation for Hierarchical Identity-Based Encryption (2007), https://eprint.iacr.org/2007/221.pdf, cryptology ePrint Archive, Paper 2007/221

[3] Alam, T.: A Reliable Communication Framework and Its Use in Internet of Things (IoT). International Journal of Scientific Research in Computer Science, Engineering and Information Technology 3(5) (2018), https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3619450

[4] Alfred J. Menezes, P.C.v.O., Vanstone, S.A.: Handbook of Applied Cryptography. CRC Press (1996)

[5] Barua, R., Dutta, R., Sarkar, P.: Extending Joux's Protocol to Multi Party Key Agreement. Cryptology ePrint Archive, Paper 2003/062 (2003), https://eprint.iacr.org/2003/062

[6] Boneh, D., Shoup, V.: A Graduate Course in Applied Cryptography (2017), https://toc.cryptobook.us/book.pdf

[7] Boneh, D., Silverberg, A.: Applications of Multilinear Forms to Cryptography. Cryptology ePrint Archive, Paper 2002/080 (2002), https://eprint.iacr.org/2002/080

[8] Cheon, J.H., Lee, C., Ryu, H.: Cryptanalysis of the New CLT Multilinear Maps. Cryptology ePrint Archive, Paper 2015/934 (2015), https://eprint.iacr.org/2015/934

[9] Cohn-Gordon, K., Cremers, C., Garratt, L., Millican, J., Milner, K.: On Ends-to-Ends Encryption: Asynchronous Group Messaging with Strong Security Guarantees. Cryptology ePrint Archive, Paper 2017/666 (2017), https://eprint.iacr.org/2017/666

[10] Connectivity Standards Alliance: Matter, https://github.com/project-chip/connectedhomeip

[11] Coron, J.S., Lepoint, T., Tibouchi, M.: Practical multilinear maps over the integers. Cryptology ePrint Archive, Paper 2013/183 (2013), https://eprint.iacr.org/2013/183

[12] Ducas, L., Pellet-Mary, A.: On the Statistical Leak of the GGH13 Multilinear Map and some Variants. Cryptology ePrint Archive, Paper 2017/482 (2017), https://eprint.iacr.org/2017/482

[13] Duttagupta, S., Singelée, D., Preneel, B.: T-HIBE: A Novel Key Establishment Solution for Decentralized, Multi-Tenant IoT Systems. In: 2022 19 IEEE Annual Consumer Communications Networking Conference (CCNC). p. 9 (2022)

[14] Farrukh, H., Ozmen, M.O., Kerem Ors, F., Celik, Z.B.: One key to rule them all: Secure group pairing for heterogeneous IoT devices. In: 2023 IEEE Symposium on Security and Privacy (SP). pp. 3026–3042 (2023)

[15] Feldman, P.: A practical scheme for non-interactive verifiable secret sharing. In: 28th Annual Symposium on Foundations of Computer Science (sfcs 1987). pp. 427–438 (1987)

[16] Fiore, D., Vasco, M.I.G., Soriente, C.: Partitioned Group Password-Based Authenticated Key Exchange (2017), https://eprint.iacr.org/2017/141.pdf, cryptology ePrint Archive, Paper 2017/141

[17] Garg, S., Gentry, C., Halevi, S.: Candidate Multilinear Maps from Ideal Lattices. In: Johansson, T., Nguyen, P.Q. (eds.) Advances in Cryptology – EUROCRYPT 2013. pp. 1–17. Springer Berlin Heidelberg, Berlin, Heidelberg (2013)

[18] 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. In: IEEE Symposium on Security and Privacy (S&P) (2018), https://ieeexplore.ieee.org/document/8418641

[19] Joux, A.: A One Round Protocol for Tripartite Diffie–Hellman. In: Bosma, W. (ed.) Algorithmic Number Theory. pp. 385–393. Springer Berlin Heidelberg, Berlin, Heidelberg (2000)

[20] Katz, J., Yung, M.: Scalable Protocols for Authenticated Group Key Exchange. Cryptology ePrint Archive, Paper 2003/171 (2003), https://eprint.iacr.org/2003/171

[21] 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. In: 28th USENIX Security Symposium. Santa Clara, CA, USA (2019), https://www.usenix.org/system/files/sec19-kumar-sam.pdf

[22] Li, X., Zeng, Q., Luo, L., Luo, T.: T2Pair: Secure and Usable Pairing for Heterogeneous IoT Devices. In: ACM SIGSAC Conference on Computer and Communications Security (CCS) (2020), https://dl.acm.org/doi/10.1145/3372297.3417286

[23] Lin, C.H., Lin, H.H., Chang, J.C.: Multiparty Key Agreement for Secure Teleconferencing. In: 2006 IEEE International Conference on Systems, Man and Cybernetics. vol. 5, pp. 3702–3707 (2006)

[24] Liu, K., Guan, J., Yao, S., Wang, L., Zhang, H.: DKGAuth: Blockchain-Assisted Distributed Key Generation and Authentication for Cross-Domain Intelligent IoT. IEEE Internet of Things Journal 11(15), 25663–25673 (2024)

[25] Lu, L., Lu, J.: A lightweight verifiable secret sharing scheme in IoTs. Cryptology ePrint Archive, Paper 2022/395 (2022), https://eprint.iacr.org/2022/395

[26] Luo, X., et al.: A Lightweight Privacy-Preserving Communication Protocol for Heterogeneous IoT Environment. IEEE Access 8, 67192–67204 (2020), https://ieeexplore.ieee.org/document/9034516

[27] 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, 67192–67204 (2020)

[28] McCune, J., Perrig, A., Reiter, M.: Seeing-is-believing: Using Camera Phones for Human-Verifiable Authentication. In: IEEE Symposium on Security and Privacy. pp. 110–124 (2005), https://ieeexplore.ieee.org/document/1425062

[29] Nyberg, K.: Fast accumulated hashing. In: Fast Software Encryption, Third International Workshop, Cambridge, UK, February 21-23, 1996, Proceedings. Lecture Notes in Computer Science, vol. 1039, pp. 83–87. Springer (1996)

[30] Zhang, T., Yi, X., Wang, R., Wang, Y., Yu, C., Lu, Y., Shi, Y.: Tap-to-Pair: Associating Wireless Devices with Synchronous Tapping. ACM Interactive, Mobile, Wearable and Ubiquitous Technologies (2018), https://dl.acm.org/doi/pdf/10.1145/3287079