Indian Institute of Engineering Science and Technology, Shibpur

(Howrah, West Bengal, India)

Blockchain-Incentivized Secure Communication for Trustless IoT Relay Networks

Submitted by:

1.	**Aninda Nath**	Roll No:	2021ETB077
2.	**Gaurav Kumar Sharma**	Roll No:	2021ETB011
3.	**Dwaipayan Biwas**	Roll No:	2021ETB021
4.	**Shibam Nath**	Roll No:	2021ETB008

Supervisor:

Dr.Niladri Das

Department of Electronics and Telecommunications Engineering

Department of Electronics and
Telecommunication Engineering
Indian Institute of Engineering
Science and Technology, Shibpur,
India - 711103

CERTIFICATE

This is to certify that we have examined the thesis entitled "**Blockchain-Incentivized Secure Communication for Trustless IoT Relay Networks** ", submitted by **Aninda Nath** (Roll Number: *2021ETB077*), **Gaurav Kumar Sharma** (Roll Number: *2021ETB011*), **Dwaipayan Biwas** (Roll Number: *2021ETB021*), **Shibam Nath** (Roll Number: *2021ETB008*) a undergraduate students of **Department of Department of Electronics and Telecommunication Engineering** in partial fulfillment for the award of degree of **Bachelor of Technology**. We hereby accord our approval of it as a study carried out and presented in a manner required for its acceptance in partial fulfillment for the under-graduate degree for which it has been submitted. The thesis has fulfilled all the requirements as per the regulations of the institute and has reached the standard needed for submission.

....
Head of Department
Dr. Tamaghna Acharya,
Dept. of E.T.C.E.,
IIEST, Shibpur.

....
Supervisor
Dr. Niladri Das,
Dept. of E.T.C.E.,
IIEST, Shibpur.

Examiners:
1. ....
2. ....
3. ....

Place: Shibpur
Date:........

Department of Department of Electronics and
Telecommunication Engineering
Indian Institute of Engineering
Science and Technology, Shibpur,
India - 711103

CERTIFICATE OF APPROVAL

The forgoing thesis report is hereby approved as a creditable study of "**Blockchain-Incentivized Secure Communication for Trustless IoT Relay Networks** "carried out and presented satisfactorily to warrant its acceptance as a prerequisite for the Degree of Bachelor of Technology. It is understood that by this approval the undersigned do not necessarily approve any statement made, opinion expressed and conclusion drawn there in but approve the progress report only for the purpose for which it is submitted.

Examiners:
1. . . . .
2. . . . .
3. . . . .

Place: Shibpur
Date:. . . . .

Acknowledgements

I would like to express my deepest gratitude to my supervisor, Dr. Niladri Das, for his invaluable guidance, support, and encouragement throughout this research. I also thank my friends, family, and the faculty members of Indian Institute of Engineering Science and Technology, Shibpur for their constant support and inspiration.

Abstract

The proliferation of Internet of Things (IoT) devices necessitates robust security mechanisms, particularly when data traverses untrusted intermediary networks. Resource constraints on IoT devices often preclude the use of computationally intensive security protocols, while ensuring reliable data relay through intermediaries presents challenges in trust and incentivization. This thesis addresses these challenges by proposing, designing, and evaluating a novel system combining a hybrid cryptographic protocol with a blockchain-based reward mechanism for secure and incentivized IoT data transmission.

The proposed hybrid protocol leverages Elliptic Curve Cryptography (ECC), specifically Elliptic Curve Diffie-Hellman (ECDH) for key exchange and Elliptic Curve Digital Signature Algorithm (ECDSA) for authentication, combined with the Advanced Encryption Standard in Galois/Counter Mode (AES-GCM) for efficient authenticated encryption of bulk data. This approach ensures end-to-end confidentiality, integrity, sender authenticity, and forward secrecy between the IoT device and a base station, even when messages are relayed via untrusted intermediaries.

To incentivize reliable data relay, a smart contract, implemented in Solidity and deployed on an Ethereum-based blockchain (simulated via Ganache), provides a trustless mechanism for rewarding intermediaries. Upon successful data reception and decryption, the receiver (either the Base Station or the IoT device, depending on the communication direction) generates cryptographic proof. This proof consists of the keccak256 hash of the original message data and an ECDSA signature of this hash, created using the receiver's designated private key. The intermediary submits this hash and signature pair to the smart contract's claimReward function. The contract verifies the signature against the hash using the pre-configured receiver's public key and checks for prior claims before automatically disbursing a predefined reward in Ether (ETH) upon successful validation, mitigating counterparty risk.

The system was implemented using Node.js, utilizing the Hardhat framework and Ethers.js library for smart contract deployment and interaction. Functional verification confirmed the end-to-end communication flow and the correct operation of the reward mechanism under various scenarios, including replay and invalid signature attempts. Security analysis demonstrates resilience against common threats like eavesdropping, tampering, and Man-in-the-Middle attacks within the defined trust model. Preliminary performance analysis quantifies cryptographic operation latencies, communication overhead, and blockchain transaction costs (gas consumption), providing insights into the system's feasibility for resource-constrained environments. This research demonstrates the viability of integrating hybrid cryptography with blockchain-based smart contracts to achieve secure, verifiable, and economically incentivized data relay in IoT ecosystems.

Contents

1 Introduction **1**
1.1 Motivation and Problem Statement . . . 1
1.1.1 The IoT Landscape and Security Imperatives . . . 1
1.1.2 Challenges in Resource-Constrained Environments . . . 1
1.1.3 The Role of Intermediaries in IoT Networks . . . 2
1.1.4 Trust and Incentivization Issues with Intermediaries . . . 2
1.1.5 Problem Definition: Secure, Verifiable, and Incentivized Data Transmission via Untrusted Intermediaries . . . 3
1.2 Proposed Solution Overview . . . 3
1.2.1 Hybrid Encryption for End-to-End Security . . . 3
1.2.2 Blockchain and Smart Contracts for Trustless Reward Verification . . . 4
1.3 Research Objectives . . . 5
1.4 Key Contributions . . . 5
1.5 Scope and Limitations . . . 6
1.6 Thesis Outline . . . 6

2 Background Technologies and Related Work **8**
2.1 Fundamentals of IoT Security . . . 8
2.1.1 Common Attack Vectors in IoT Communication . . . 8
2.1.2 Security Limitations of Standard Protocols (e.g., TLS/DTLS) in Constrained Environments . . . 9
2.2 Cryptographic Primitives Employed . . . 10
2.2.1 Asymmetric Cryptography: Elliptic Curve Cryptography (ECC) . . . 11
2.2.2 Symmetric Cryptography: Advanced Encryption Standard (AES) . . . 15
2.2.3 Key Derivation Functions (KDF) . . . 17
2.3 Blockchain and Smart Contract Fundamentals . . . 20
2.3.1 Distributed Ledger Technology (DLT) Concepts . . . 20
2.3.2 Ethereum Blockchain Platform . . . 21
2.3.3 Smart Contracts: Principles and Lifecycle . . . 22
2.4 Related Work . . . 23
2.4.1 Existing Secure Communication Protocols for IoT . . . 23
2.4.2 Blockchain Applications in IoT . . . 24
2.4.3 Incentive Mechanisms for Network Participation (Cryptoeconomics) . . . 25
2.4.4 Analysis of Existing Solutions and Identification of Research Gaps . . . 26

3 System Design and Architecture **27**
3.1 Overall System Architecture . . . 27

3.1.1 Component Roles and Responsibilities . . . 27
3.1.2 Trust Assumptions and Threat Model . . . 29
3.2 Hybrid Encryption Protocol Design . . . 29
3.2.1 Key Management Strategy . . . 29
3.2.2 Uplink Communication Protocol (IoT -> Base Station) . . . 30
3.2.3 Downlink Communication Protocol (Base Station -> IoT) . . . 31
3.3 Blockchain-Based Reward Mechanism . . . 33
3.3.1 MessageReward Smart Contract Design . . . 33
3.3.2 Off-Chain Proof Generation (Receiver: Base Station/IoT Device) . . . 35
3.3.3 Interaction Flow for Reward Claim . . . 36

4 Implementation Details **37**
4.1 Development Environment and Technology Stack . . . 37
4.2 Component Implementation (`Aninda001/IoT-blockchain` repository) . 38
4.3 Smart Contract Deployment and Initialization . . . 38

5 Design Analysis and Discussion **40**
5.1 Introduction . . . 40
5.2 Security Analysis . . . 40
5.2.1 Confidentiality Analysis (Against Eavesdropping) . . . 40
5.2.2 Integrity Analysis (Against Tampering - AES-GCM Tag) . . . 41
5.2.3 Authenticity Analysis (ECDSA Signatures) . . . 41
5.2.4 Replay Attack Resistance (Ephemeral Keys/Nonces & Contract State) . 41
5.2.5 Man-in-the-Middle (MitM) Attack Resilience . . . 41
5.2.6 Smart Contract Security Considerations . . . 42
5.3 Theoretical Performance Analysis . . . 42
5.3.1 Analysis of Cryptographic Operation Costs (Relative Intensity) . . . 42
5.3.2 Communication Overhead Calculation . . . 43
5.3.3 Estimated Blockchain Transaction Costs . . . 43
5.3.4 Performance Implications for Resource-Constrained Devices . . . 44
5.4 Discussion . . . 44
5.4.1 Interpretation of Analytical Findings . . . 44
5.4.2 Limitations of the Study . . . 45
5.4.3 Future Work and Enhancements . . . 45

6 Conclusion **46**
6.1 Summary of Research and Contributions . . . 46
6.2 Concluding Remarks on the Proposed System . . . 46

References **48**

List of Figures

2.1 Elliptic Curve Point Addition 12
2.2 ECDH Key Exchange Flow 13
2.3 ECDSA Signing and Verification Process 14
2.4 AES-GCM Encryption Process 18
2.5 HKDF Extract-then-Expand Process 19

3.1 Detailed System Architecture 28
3.2 Uplink Sequence Diagram 30
3.3 Downlink Sequence Diagram 32
3.4 Flowchart for claimReward Logic 35

4.1 Deployment & Funding Sequence Diagram 39

List of Tables

2.1 Comparison of Cryptographic Key Sizes and Security Levels 15
2.2 Comparison of Related Works/Protocols 26

3.1 Uplink Data Packet Structure 31

5.1 Theoretical Communication Overhead Analysis 43
5.2 Estimated Relative Gas Cost Factors for claimReward Operations 44

Chapter 1

Introduction

1.1 Motivation and Problem Statement

The dawn of the Internet of Things (IoT) era has ushered in an unprecedented level of connectivity, weaving a complex fabric of interconnected physical objects – sensors, actuators, appliances, vehicles, and countless other devices – capable of collecting, processing, and exchanging vast amounts of data. This technological paradigm shift promises transformative advancements across diverse sectors, including smart cities, industrial automation (IIoT), environmental monitoring, intelligent transportation systems, personalized healthcare, and precision agriculture [1]. The potential benefits are immense, ranging from enhanced operational efficiency and resource optimization to improved quality of life and novel service creation.

1.1.1 The IoT Landscape and Security Imperatives

The scale of IoT deployment is staggering, with projections estimating tens of billions of connected devices globally within the next few years [2]. These devices often operate autonomously, interacting directly with the physical environment and generating data that can be highly sensitive, proprietary, or critical for operational safety. Consequently, the security of IoT systems is not merely a technical requirement but a fundamental imperative. Breaches in IoT security can have far-reaching consequences, extending beyond data loss or privacy violations to encompass physical damage, disruption of critical infrastructure, economic losses, and even threats to human safety [3]. Therefore, establishing robust security postures encompassing confidentiality (preventing unauthorized data disclosure), integrity (ensuring data has not been tampered with), availability (guaranteeing service accessibility), and authenticity (verifying the identity of communicating parties) is paramount for the successful and trustworthy adoption of IoT technologies.

1.1.2 Challenges in Resource-Constrained Environments

A significant portion of the IoT device ecosystem consists of resource-constrained embedded systems, often characterized by limitations in:

- **Computational Power**: Low-frequency microcontrollers (MCUs) with limited processing capabilities (measured in MHz rather than GHz).

- **Memory:** Scarce Random Access Memory (RAM, often in Kilobytes) and limited non-volatile storage (Flash memory, in Kilobytes or Megabytes).
- **Energy Budget:** Devices frequently operate on batteries or rely on energy harvesting, demanding ultra-low power consumption and imposing strict limits on energy-intensive operations.
- **Network Bandwidth:** Communication often occurs over Low-Power Wide-Area Networks (LPWANs) like LoRaWAN or NB-IoT, which offer long range but low data rates and strict duty cycle limitations.

These constraints render many traditional security protocols, such as Transport Layer Security (TLS) or Datagram TLS (DTLS) in their standard configurations, impractical or inefficient. The computational overhead associated with complex cryptographic handshakes (particularly public-key operations like RSA or full ECDH suites), large code footprints, significant memory requirements for session state and certificates, and the energy consumed during prolonged communication cycles pose significant challenges [4]. This necessitates the development and adoption of lightweight cryptographic primitives and security protocols specifically tailored for the operational realities of resource-constrained IoT environments.

1.1.3 The Role of Intermediaries in IoT Networks

Many IoT deployment architectures rely on intermediary nodes to facilitate communication between end devices and backend systems (often referred to as base stations or cloud platforms). These intermediaries serve various crucial functions:

- **Range Extension:** Bridging the gap between low-power, short-range devices and longer-range backhaul networks (e.g., LoRaWAN gateways collecting data from numerous end nodes).
- **Protocol Translation:** Converting between different network protocols (e.g., translating CoAP messages to HTTP).
- **Data Aggregation:** Performing preliminary aggregation or filtering of data from multiple devices before forwarding.
- **Network Management:** Assisting in managing device connectivity and network traffic.

1.1.4 Trust and Incentivization Issues with Intermediaries

While essential for connectivity, these intermediaries often represent a critical point of vulnerability, particularly if they are not owned or operated by the same entity controlling the end devices or the base station. The intermediary node lies directly on the data path, introducing significant trust assumptions:

- **Potential for Eavesdropping:** If data is not encrypted end-to-end between the IoT device and the base station, the intermediary can potentially access sensitive information.
- Data Tampering Risk: An untrusted intermediary could modify data in transit if adequate integrity protection mechanisms are not implemented end-to-end.
- **Denial of Service (DoS):** The intermediary might selectively or entirely drop packets, disrupting communication without the end device or base station necessarily being aware.

- **Lack of Verifiable Incentive:** In scenarios where intermediaries are operated by third parties (e.g., community LoRaWAN gateways, commercial network providers), their motivation to reliably and promptly relay data might be purely contractual or based on goodwill. There is often no built-in, technically verifiable mechanism to ensure they perform their function correctly or to reward them directly and proportionally for successful data relay. This lack of verifiable incentivization can lead to unreliable service or create barriers for deploying open, multi-party IoT networks.

This reliance on potentially untrusted and un-incentivized intermediaries creates a significant "trust gap" in many IoT architectures.

1.1.5 Problem Definition: Secure, Verifiable, and Incentivized Data Transmission via Untrusted Intermediaries

Based on the preceding motivations and challenges, the central problem addressed in this thesis can be defined as follows:

How can end-to-end secure (confidential, integrity-protected, authenticated) communication be established between resource-constrained IoT devices and a base station, when data must be relayed through potentially untrusted intermediaries, while simultaneously providing a trustless, verifiable, and automated mechanism to incentivize these intermediaries for reliable data transmission?

This problem necessitates a holistic solution that integrates lightweight, robust cryptographic techniques suitable for constrained devices with a transparent and automated incentive system that does not rely on traditional, trust-based contractual agreements. The solution must ensure that security guarantees hold even if the intermediary is compromised or uncooperative, and that rewards are disbursed only for demonstrably successful data relay actions, directly linking the incentive to the desired outcome.

1.2 Proposed Solution Overview

To address the multifaceted problem of achieving secure, verifiable, and incentivized data transmission via untrusted intermediaries in resource-constrained IoT environments, this thesis proposes and investigates a system built upon two core technological pillars: a hybrid cryptographic protocol for robust end-to-end security and a blockchain-based smart contract for implementing a trustless, automated reward mechanism.

1.2.1 Hybrid Encryption for End-to-End Security

Recognizing the performance limitations of purely asymmetric cryptography and the key distribution challenges of purely symmetric cryptography in this context, a hybrid approach is adopted. This approach leverages the strengths of both paradigms:

- **Asymmetric Cryptography (ECC):** Elliptic Curve Diffie-Hellman (ECDH) is employed for establishing a shared secret between the IoT end device and the base station for each communication session. This utilizes pre-shared static public keys for authentication during the key exchange and incorporates ephemeral keys generated per-session to achieve Perfect Forward Secrecy (PFS). PFS ensures that the compromise of long-term static keys does not compromise the confidentiality of past communication sessions. Furthermore, the Elliptic Curve Digital Signature Algorithm (ECDSA) is used by the sender (IoT device or base station) to sign the encrypted payload, providing data origin authentication and integrity assurance against tampering by the intermediary or other attackers.

- **Symmetric Cryptography (AES-GCM):** The shared secret derived via ECDH is securely transformed using a Key Derivation Function (HKDF) into symmetric keys (specifically, a 128-bit AES key and a unique nonce). The actual message data is then encrypted using the highly efficient Advanced Encryption Standard (AES) operating in Galois/-Counter Mode (GCM). AES-GCM is chosen not only for its speed, making it suitable for constrained devices, but also because it provides Authenticated Encryption with Associated Data (AEAD). This means it simultaneously ensures both confidentiality and integrity/authenticity of the encrypted data through an integrated Message Authentication Code (MAC), often referred to as an authentication tag.

This hybrid design ensures that the bulk data encryption is performed using fast symmetric algorithms, minimizing the load on the IoT device, while the critical key establishment and authentication rely on the robust security properties of ECC. Crucially, the entire encryption/decryption process occurs only at the endpoints (IoT device and base station), rendering the intermediary incapable of accessing the plaintext data, regardless of its trustworthiness.

1.2.2 Blockchain and Smart Contracts for Trustless Reward Verification

To address the challenge of reliably incentivizing potentially untrusted intermediaries, the proposed system integrates a decentralized reward mechanism built upon blockchain technology, specifically utilizing an Ethereum-compatible smart contract.

- **Decentralized Trust:** Blockchain technology provides a distributed, immutable ledger, eliminating the need for a central trusted authority to manage rewards. Transactions and contract state changes are verified by the network consensus mechanism, providing transparency and tamper-resistance.
- **Smart Contract Automation:** A smart contract, implemented in the Solidity language and deployed on the blockchain (simulated using Ganache for this research), codifies the rules for reward disbursement. This contract holds the reward funds (e.g., in Ether) and contains the logic to verify proof of successful data relay.
- **Verifiable Proof of Relay:** Upon successful reception and validation of data, the receiver (base station or IoT device) generates cryptographic proof. This proof consists of the keccak256 hash of the received message and an ECDSA signature of this hash, created using the receiver's private key.
- **Automated, Trustless Disbursement:** The intermediary submits this proof (hash and signature) to the smart contract's `claimReward` function. The contract autonomously verifies:
  - The validity of the signature against the hash using the receiver's known public key (stored within the contract).
  - That the reward for this specific hash has not already been claimed (preventing double-spending).
  - That sufficient funds are available in the contract.

If all checks pass, the contract automatically transfers the predefined reward amount directly to the intermediary's blockchain address (`msg.sender`). This process eliminates counterparty risk and the need for manual invoicing or reconciliation, directly linking the reward to cryptographically verifiable proof of service.

This blockchain integration provides a transparent, automated, and secure framework for incentivizing intermediaries, fostering participation in open IoT relay networks without requiring pre-existing trust relationships.

1.3 Research Objectives

The primary objectives of this research are:

1. **Design:** To design a hybrid cryptographic protocol optimized for secure end-to-end communication between resource-constrained IoT devices and a base station, resilient to untrusted intermediaries.
2. **Integrate:** To architect and design a blockchain-based reward system using a smart contract to provide verifiable, automated incentives for data relay by intermediaries.
3. **Implement:** To implement a functional prototype of the integrated system, including the IoT device, intermediary, base station logic (using Node.js), and the smart contract (using Solidity on the Hardhat/Ganache environment).
4. **Verify:** To functionally verify the correctness of the end-to-end communication protocol and the blockchain reward mechanism through targeted test cases.
5. **Analyze:** To perform a security analysis of the proposed system against relevant threats and evaluate its performance characteristics, including cryptographic operation latency, communication overhead, and blockchain transaction costs within the simulated environment.

1.4 Key Contributions

This thesis makes the following key contributions:

- **Novel System Integration:** Presents a cohesive system architecture integrating lightweight hybrid cryptography (ECDH/AES-GCM/ECDSA) with a blockchain-based smart contract (`MessageReward`) specifically designed for incentivized, secure IoT data relay via untrusted intermediaries.
- **Trustless Incentive Mechanism Design:** Details the design and implementation of a smart contract-based reward mechanism that utilizes receiver-generated cryptographic proof (keccak256 hash + ECDSA signature) for verifiable and automated reward disbursement.
- **Implementation and Simulation:** Provides a practical implementation and simulation of the complete system using contemporary tools (Node.js, Hardhat, Ethers.js, Solidity, Ganache), serving as a blueprint and testbed for the proposed concepts.
- **Security and Performance Evaluation:** Offers an analysis of the system's security properties within its defined threat model and an evaluation of its performance over-

head (cryptographic, communication, blockchain transaction costs) in a simulated context, providing insights into its practical viability.

1.5 Scope and Limitations

The scope of this research is defined as follows:

- **Focus:** The primary focus is on the design, implementation, and evaluation of the cryptographic protocol and the blockchain-based incentive mechanism in a simulated environment.
- **Cryptography:** Utilizes specific cryptographic primitives (NIST P-256 curve, AES-128-GCM, HKDF-SHA256, ECDSA-SHA256, keccak256). Analysis of alternative primitives is outside the scope.
- **Environment:** The implementation and evaluation are conducted within a simulated network environment using Node.js and a local Ganache blockchain instance. Testing on real-world, resource-constrained hardware or large-scale network deployments is not included.
- **Key Management:** Assumes secure pre-distribution or configuration of long-term static public keys for the IoT device and base station. Detailed protocols for secure key provisioning and rotation are outside the scope.
- **Blockchain:** Focuses on an Ethereum-like (EVM-compatible) blockchain. Exploration of other DLTs or Layer 2 scaling solutions is not covered.
- **Security Analysis:** Addresses confidentiality, integrity, authenticity, replay resistance, and MitM threats within the defined model. Excludes analysis of physical security, side-channel attacks, advanced smart contract vulnerabilities (e.g., reentrancy beyond basic checks), or denial-of-service attacks against the blockchain network itself.
- **Intermediary Model:** Assumes intermediaries are rational actors motivated by the reward but potentially untrustworthy regarding data handling. Complex collusion scenarios are not deeply analyzed.

1.6 Thesis Outline

The remainder of this thesis is structured as follows:

- **Chapter 2:** Background Technologies and Related Work: Provides a detailed overview of the fundamental concepts in IoT security, the cryptographic primitives used (ECC, AES-GCM, HKDF), blockchain and smart contract technologies (Ethereum, Solidity), and reviews existing related work in secure IoT communication and blockchain-based incentivization, identifying the research gap this work addresses.
- **Chapter 3:** System Design and Architecture: Presents the detailed design of the proposed system, including the overall architecture, the step-by-step specification of the hybrid encryption protocol (uplink and downlink), the design of the `MessageReward` smart contract, and the interaction flow for the reward claim mechanism.

- **Chapter 4:** Implementation Details: Describes the practical implementation of the system components, covering the technology stack, software modules (IoT device, intermediary, base station), smart contract code structure, cryptographic utilities, configuration management, and the deployment process using Hardhat Ignition.
- **Chapter 5:** Design Analysis and Discussion: Theoretically analyzes the system's security and performance, discusses the findings, trade-offs, and limitations (notably lacking empirical tests), and suggests future research directions.
- **Chapter 6:** Conclusion: Summarizes the key findings and contributions of the research and offers concluding remarks on the potential impact of the proposed system.

Chapter 2

Background Technologies and Related Work

This chapter provides a comprehensive background on the core technologies and concepts underpinning the proposed system. It begins by exploring the fundamental security challenges inherent in the Internet of Things (IoT) landscape. Subsequently, it delves into the specific cryptographic primitives employed in the hybrid encryption protocol, detailing their mechanisms and properties. The chapter then introduces the relevant concepts of blockchain technology and smart contracts, focusing on the Ethereum platform. Finally, it reviews pertinent related work in the fields of secure IoT communication protocols and blockchain-based incentive systems, establishing the context and novelty of the research presented in this thesis.

2.1 Fundamentals of IoT Security

The unique characteristics of IoT systems – encompassing massive scale, device heterogeneity, resource constraints, direct interaction with the physical world, and complex network topologies often involving intermediaries – introduce a distinct set of security challenges compared to traditional IT environments. Securing these systems requires a holistic approach, but at the core lies the need to protect data communication between devices and backend systems. Failure to secure these communication channels can expose the entire system to various attacks.

2.1.1 Common Attack Vectors in IoT Communication

Several attack vectors specifically target the communication links within IoT networks. Understanding these threats is crucial for designing effective countermeasures. Key attack vectors relevant to this work include:

- **Eavesdropping (Sniffing):** Due to the prevalence of wireless communication media (e.g., Wi-Fi, Bluetooth LE, LoRaWAN, Zigbee) in IoT deployments, communication signals are often broadcast over the air. Attackers equipped with appropriate receivers can passively intercept these transmissions without the legitimate parties' knowledge [5]. If the communication is not encrypted, the attacker gains direct access to potentially sensitive plaintext data, violating confidentiality. This could include sensor readings, control commands, device identifiers, or even cryptographic keying material exchanged insecurely.

- **Man-in-the-Middle (MitM) Attacks:** In a MitM attack, the adversary positions themselves strategically on the communication path between two legitimate parties (e.g., between an IoT device and an intermediary, or between an intermediary and a base station). The attacker intercepts messages from the sender, potentially reads or modifies them, and then forwards them (possibly altered) to the intended receiver. The legitimate parties may remain unaware that their communication is being relayed through a malicious entity [6]. This attack directly threatens confidentiality (if the attacker can decrypt or sees plaintext), integrity (if the attacker modifies messages), and authenticity (as parties might be communicating with the attacker unknowingly). The presence of required intermediaries in many IoT architectures makes this threat particularly pertinent, as a compromised or malicious intermediary inherently acts as a MitM.

- **Replay Attacks:** A replay attack involves an adversary capturing legitimate, encrypted messages transmitted over the network and re-transmitting them at a later time [7]. Even if the attacker cannot decrypt the message content (due to encryption), replaying a valid, previously authenticated message can cause unintended consequences. For example, replaying a command to "unlock a door" or "turn on a valve" could lead to physical security breaches or operational failures. Replaying sensor data might disrupt monitoring systems or trigger incorrect responses. This attack violates the freshness property of communication and can compromise system integrity. Effective defenses typically involve incorporating unique, non-repeating elements like nonces, timestamps, or sequence numbers within authenticated messages.

- **Data Tampering (Modification):** Closely related to MitM attacks, data tampering involves the unauthorized modification of data in transit. An attacker might intercept a message, alter its payload (e.g., change a sensor reading from "normal" to "critical," modify a control command's parameters), and then forward the altered message to the receiver. If the receiver lacks a mechanism to verify the data's integrity, they might accept and act upon the falsified information [8]. This directly violates data integrity and can lead to severe system malfunctions, incorrect decision-making, or unsafe operating conditions. Robust integrity protection typically relies on cryptographic Message Authentication Codes (MACs) or digital signatures.

While other attacks like Denial of Service (DoS), physical device tampering, firmware manipulation, and side-channel attacks also pose significant threats to IoT systems, this thesis primarily focuses on securing the communication channel against eavesdropping, MitM, replay, and tampering through cryptographic means at the protocol level. Addressing physical security or application-level vulnerabilities falls outside the defined scope (as mentioned in Section 1.5).

2.1.2 Security Limitations of Standard Protocols (e.g., TLS/DTLS) in Constrained Environments

Transport Layer Security (TLS) and its datagram-oriented counterpart, Datagram TLS (DTLS), are the bedrock protocols for securing communications over TCP/IP networks, including the internet [9]. They provide confidentiality, integrity, and authentication using a combination of symmetric and asymmetric cryptography. However, deploying standard TLS/DTLS stacks on highly resource-constrained IoT devices faces significant hurdles:

- **Computational Overhead:** The TLS/DTLS handshake phase, particularly the public-key operations required for key exchange (e.g., RSA, Diffie-Hellman, ECDH suites) and certificate signature verification, is computationally intensive [10]. Low-power MCUs may take seconds or even tens of seconds to complete a full handshake, consuming considerable CPU cycles and impacting real-time responsiveness. While Elliptic Curve Cryptography (ECC) variants are more efficient than RSA or traditional Diffie-Hellman for the same security level, the overhead can still be prohibitive for the most constrained devices.

- **Memory Footprint:** Implementing a full TLS/DTLS stack requires substantial memory resources. This includes RAM for storing session state (keys, sequence numbers, buffers), cryptographic context, and potentially large message buffers during the handshake. Furthermore, the code size (Flash memory) for the protocol stack itself, along with the underlying cryptographic libraries, can easily exceed the available storage on many low-end IoT devices [11]. The need to store and process X.509 digital certificates, often including intermediate certificate authorities (CAs) to form a trust chain, further exacerbates memory requirements.
- **Energy Consumption:** The combination of high computational load during the handshake and potentially increased data transmission (due to handshake message sizes) translates directly into higher energy consumption [12]. For battery-powered devices designed to operate for months or years on a single charge, the energy cost of establishing frequent TLS/DTLS sessions can drastically reduce operational lifetime, rendering it impractical for many use cases, especially those requiring frequent, short data bursts.
- **Communication Overhead:** The TLS/DTLS handshake involves multiple message exchanges (round-trips) between the client and server. These handshake messages, especially those carrying certificates or long lists of cipher suites, can be significantly larger than the actual application data payload being transmitted [13]. On low-bandwidth, high-latency networks like LPWANs (e.g., LoRaWAN, NB-IoT), which often have strict limitations on packet size (payload fragmentation) and transmission frequency (duty cycles), the overhead associated with establishing a TLS/DTLS session can be unacceptable, consuming excessive bandwidth and potentially violating network regulations.
- **Complexity and Management:** Implementing, configuring, and managing a full TLS/DTLS stack, including robust certificate validation (checking expiration, revocation status via CRLs or OCSP), is complex. Securely provisioning and updating certificates and trust anchors on potentially millions of deployed devices presents significant logistical challenges.

These limitations highlight the need for security solutions specifically designed or adapted for the constraints of the IoT environment. While efforts exist to optimize TLS/DTLS (e.g., session resumption, connection ID in DTLS 1.3, use of pre-shared keys or raw public keys instead of full certificates), alternative approaches like the hybrid protocol proposed in this thesis aim to provide comparable security guarantees with significantly reduced overhead by carefully selecting lightweight primitives and minimizing handshake complexity.

2.2 Cryptographic Primitives Employed

The security of the proposed hybrid protocol hinges on the careful selection and correct application of robust cryptographic primitives. This section details the asymmetric (ECC), symmetric (AES-GCM), and key derivation (HKDF) techniques utilized, explaining their operational principles and mathematical foundations.

2.2.1 Asymmetric Cryptography: Elliptic Curve Cryptography (ECC)

Elliptic Curve Cryptography (ECC) provides public-key cryptographic mechanisms based on the algebraic structure of elliptic curves over finite fields. Compared to traditional public-key systems like RSA, ECC offers equivalent security levels with significantly smaller key sizes, resulting in reduced computational overhead, lower memory requirements, and decreased bandwidth consumption. These characteristics make ECC particularly well-suited for resource-constrained IoT environments [14].

Mathematical Foundations of ECC

An elliptic curve over a finite field defines a set of points that satisfy a specific equation. For cryptographic purposes, we typically consider curves over prime fields (denoted F_p, where p is a large prime). A non-supersingular elliptic curve E over F_p (where $p > 3$) is defined by the set of points (x, y) satisfying the short Weierstrass equation:

$$y^2 \equiv x^3 + ax + b \pmod{p} \tag{2.1}$$

where $x, y, a, b \in F_p$, and the discriminant $\Delta = -16(4a^3+27b^2) \not\equiv 0 \pmod{p}$ (to avoid singularities). The set of points on the curve $E(F_p)$ includes all pairs (x, y) satisfying Equation 2.1, plus a special point called the point at infinity, denoted $\mathcal{O}$.

A crucial property of elliptic curves is that an abelian group structure can be defined over the set of points $E(F_p)$. This group operation, often visualized as "point addition," has specific geometric interpretations and algebraic formulas:

- **Identity Element:** The point at infinity $\mathcal{O}$ serves as the identity element: $P + \mathcal{O} = \mathcal{O} + P = P$ for any point P on the curve.
- **Negation:** The negative of a point $P = (x, y)$ is $-P = (x, -y \pmod{p})$. Note that $P + (-P) = \mathcal{O}$.
- **Point Addition:** For two distinct points $P = (x_1, y_1)$ and $Q = (x_2, y_2)$ where $P \neq -Q$, their sum $R = P + Q = (x_3, y_3)$ is calculated as:

$$\begin{aligned}\lambda &= (y_2 - y_1)/(x_2 - x_1) \pmod{p} \quad \text{(slope of the line through P and Q)}\\ x_3 &= \lambda^2 - x_1 - x_2 \pmod{p}\\ y_3 &= \lambda(x_1 - x_3) - y_1 \pmod{p}\end{aligned}$$

- **Point Doubling:** For a point $P = (x_1, y_1)$, where $y_1 \neq 0$, the point $R = 2P = P + P = (x_3, y_3)$ is calculated as:

$$\begin{aligned}\lambda &= (3x_1^2 + a)/(2y_1) \pmod{p} \quad \text{(slope of the tangent line at P)}\\ x_3 &= \lambda^2 - 2x_1 \pmod{p}\\ y_3 &= \lambda(x_1 - x_3) - y_1 \pmod{p}\end{aligned}$$

Scalar Multiplication: The core operation underlying ECC's security is scalar multiplication. Given an integer d (the scalar) and a point P on the curve, scalar multiplication computes

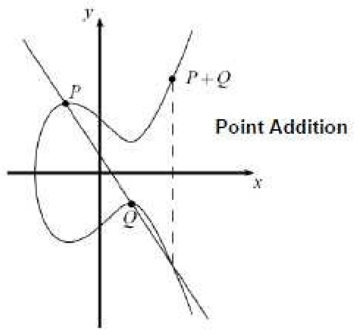

Figure 2.1: Elliptic Curve Point Addition

$Q = dP$, which means adding the point P to itself d times ($P + P + \cdots + P$). This operation can be performed efficiently using algorithms like the double-and-add method.

Elliptic Curve Discrete Logarithm Problem (ECDLP): The security of ECC relies on the presumed computational hardness of the ECDLP. Given a base point G of large prime order n on the curve and another point Q which is known to be a scalar multiple of G (i.e., $Q = dG$), the ECDLP is the problem of finding the integer scalar d. While computing Q from d and G is efficient (scalar multiplication), computing d from Q and G is believed to be computationally infeasible for well-chosen curves and sufficiently large n, given current classical and quantum computing knowledge (though quantum computers pose a future threat via Shor's algorithm) [15].

Elliptic Curve Diffie-Hellman (ECDH) Key Exchange

ECDH is a key agreement protocol that allows two parties (say, Alice and Bob), each having an ECC public-private key pair, to establish a shared secret over an insecure channel without prior shared information, except for agreed-upon domain parameters (the curve E, the base point G, its order n).

Domain Parameters: (E, G, n, h) - Curve, Base Point, Order of G, Cofactor. These must be agreed upon beforehand.

Key Pairs:

- Alice: Private key d_A (a randomly chosen integer $1 \leq d_A \leq n-1$), Public key $Q_A = d_A G$
- Bob: Private key d_B (a randomly chosen integer $1 \leq d_B \leq n-1$), Public key $Q_B = d_B G$

Protocol Steps:

1. Alice generates her key pair (d_A, Q_A). Bob generates his key pair (d_B, Q_B).
2. Alice sends her public key Q_A to Bob.
3. Bob sends his public key Q_B to Alice.
4. Alice computes the shared secret $S = d_A Q_B = d_A(d_B G) = (d_A d_B)G$.
5. Bob computes the shared secret $S = d_B Q_A = d_B(d_A G) = (d_B d_A)G$.

Both Alice and Bob arrive at the same shared secret point S. Typically, the x-coordinate of the point S, denoted x_S, is used as the raw shared secret material, which is then usually passed through a Key Derivation Function (KDF) like HKDF (see Section 2.2.3) to produce symmetric keys of the required length and cryptographic strength.

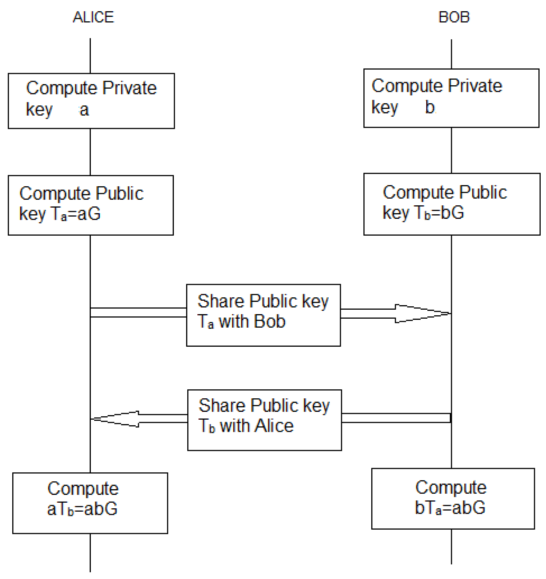

Figure 2.2: ECDH Key Exchange Flow

Security: ECDH security relies on the hardness of the ECDLP. An eavesdropper observing Q_A and Q_B cannot feasibly compute d_A or d_B, and thus cannot compute the shared secret S. ECDH itself does not provide authentication; it is vulnerable to MitM attacks if the public keys Q_A and Q_B are not authenticated (e.g., via pre-shared keys or certificates). In this thesis, authentication is handled separately by pre-shared static keys and ECDSA signatures. The use of ephemeral ECDH keys provides Perfect Forward Secrecy (PFS).

Elliptic Curve Digital Signature Algorithm (ECDSA)

ECDSA is a digital signature algorithm that uses ECC to provide data integrity, authentication, and non-repudiation.

Domain Parameters: (E, G, n, h) - Same as ECDH.

Key Pair:

- Signer: Private key d (a random integer $1 \leq d \leq n-1$), Public key $Q = dG$.

Signing Process: To sign a message m:

1. Compute the hash of the message: $e = \text{HASH}(m)$. HASH is a cryptographic hash function (e.g., SHA-256). Let z be the L_n leftmost bits of e, where L_n is the bit length of the group order n.
2. Generate a cryptographically secure random integer k such that $1 \leq k \leq n-1$. This value k must be unique and secret for each signature. Nonce reuse is catastrophic for ECDSA security.
3. Compute the curve point $(x_1, y_1) = kG$.

4. Compute $r = x_1 \pmod{n}$. If $r = 0$, go back to step 2.
5. Compute $s = k^{-1}(z + rd) \pmod{n}$. k^{-1} is the modular multiplicative inverse of k modulo n. If $s = 0$, go back to step 2.
6. The signature is the pair (r, s).

Verification Process: To verify a signature (r, s) on a message m using the signer's public key Q:

1. Verify that r and s are integers in the interval $[1, n - 1]$. If not, the signature is invalid.
2. Compute the hash $e = \text{HASH}(m)$ and let z be the L_n leftmost bits of e.
3. Compute $w = s^{-1} \pmod{n}$.
4. Compute $u_1 = zw \pmod{n}$.
5. Compute $u_2 = rw \pmod{n}$.
6. Compute the curve point $(x_1, y_1) = u_1G + u_2Q$. If $(x_1, y_1) = \mathcal{O}$ (point at infinity), the signature is invalid.
7. Compute $v = x_1 \pmod{n}$.
8. The signature is valid if and only if $v = r$.

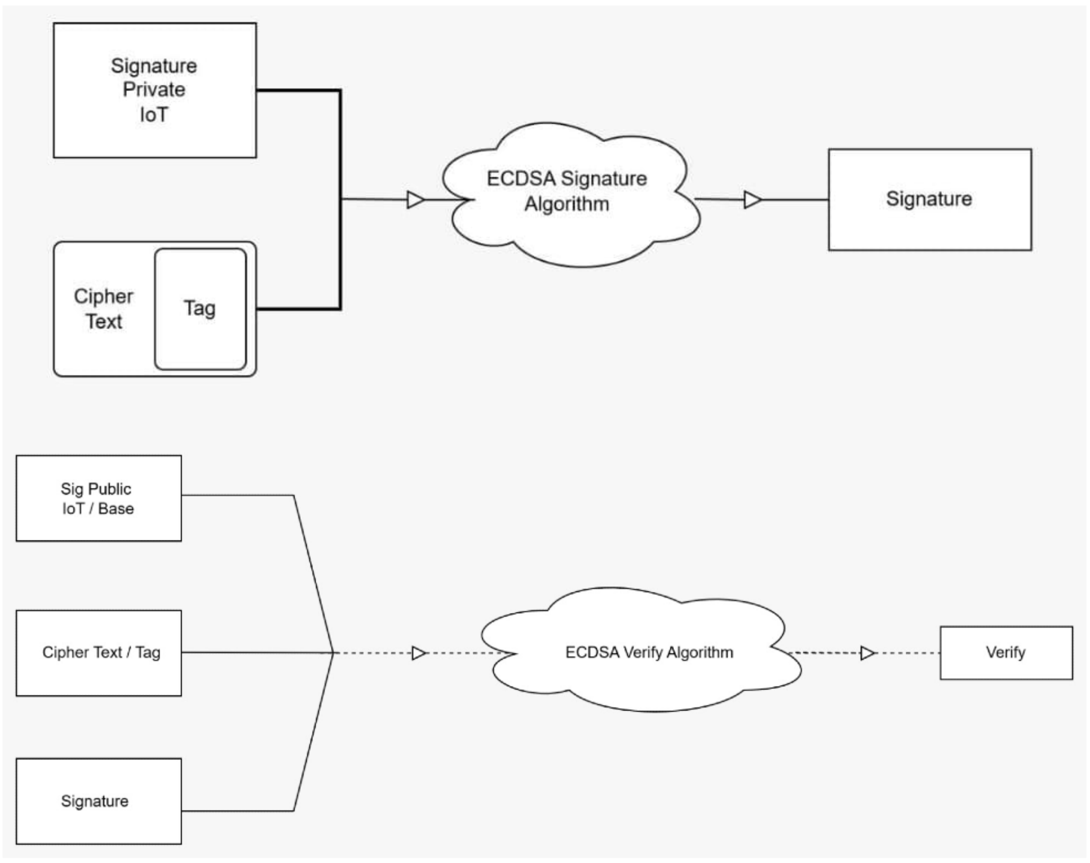

Figure 2.3: ECDSA Signing and Verification Process

Curve Selection Justification (NIST P-256 / secp256r1)

The specific elliptic curve chosen for cryptographic operations significantly impacts both security and performance. This project utilizes the NIST P-256 curve, also known under the SECG standard name secp256r1 [16]. The justification for this choice includes:

- **Security Level:** P-256 provides approximately 128 bits of security. This means that the best-known attack against ECDLP on this curve requires roughly 2^{128} operations, which is considered computationally infeasible with current technology and aligns with the security level provided by AES-128. This level is widely accepted as sufficient for protecting sensitive information for the foreseeable future.

Table 2.1: Comparison of Cryptographic Key Sizes and Security Levels

Algorithm	Key Size (bits)	Symmetric Security Level (bits)
AES	128	128
AES	256	256
RSA	2048	$\sim$112
RSA	3072	$\sim$128
ECC (P-256)	256	$\sim$128
ECC (P-384)	384	$\sim$192
ECC (P-521)	521	$\sim$256

- **Performance:** Compared to higher security curves like P-384 or P-521, operations on P-256 are significantly faster, making it a better fit for resource-constrained IoT devices where computational latency and energy consumption are critical concerns. While curves like Curve25519 might offer slightly better performance or side-channel resistance properties in some implementations, P-256 represents a well-established balance.
- **Standardization and Interoperability:** P-256 is standardized by NIST (FIPS 186-4) and widely adopted in numerous security protocols and standards (e.g., TLS, various PKI implementations). This ensures broad interoperability and availability of well-vetted cryptographic libraries across different platforms.
- **Library Support:** Robust and optimized implementations of P-256 operations are readily available in common cryptographic libraries, including Node.js's built-in crypto module, simplifying development and reducing the risk of implementation errors.

While concerns about the provenance of NIST curve generation parameters exist in some academic circles [17], P-256 remains a widely trusted and deployed standard for the 128-bit security level.

2.2.2 Symmetric Cryptography: Advanced Encryption Standard (AES)

AES is a symmetric block cipher standardized by NIST (FIPS 197) [18]. It operates on fixed-size blocks of data (128 bits) and uses keys of different lengths (128, 192, or 256 bits). AES has become the global standard for symmetric encryption due to its security, performance, and efficiency across various hardware and software platforms.

AES Algorithm Overview

AES is based on the Rijndael cipher and employs a Substitution-Permutation Network (SPN) structure. It iteratively applies a series of transformations, called rounds, to a 128-bit block of data (represented as a 4x4 matrix of bytes called the state). The number of rounds depends on the key size (10 rounds for 128-bit keys, 12 for 192-bit, 14 for 256-bit). Each round (except the final one) consists of four main transformations:

- **SubBytes:** A non-linear byte substitution using a fixed lookup table (S-box) applied independently to each byte of the state. Provides confusion.
- **ShiftRows:** A permutation step where the bytes in the last three rows of the state are cyclically shifted by different offsets. Provides diffusion.
- **MixColumns:** A transformation operating on the columns of the state, combining the four bytes in each column using multiplication and addition in the finite field $\mathbb{GF}(2^8)$. Provides further diffusion. (This step is omitted in the final round).
- **AddRoundKey:** The current round key (derived from the main AES key via a key schedule algorithm) is XORed with the state.

The security of AES relies on the complex interaction of these transformations over multiple rounds, making it resistant to various cryptanalytic attacks like linear and differential cryptanalysis when used with appropriate key lengths and modes of operation.

Galois/Counter Mode (GCM) for Authenticated Encryption (AEAD)

Simply encrypting data with a block cipher like AES in basic modes (e.g., ECB, CBC) does not guarantee data integrity or authenticity. An attacker could potentially modify the ciphertext without detection. Authenticated Encryption with Associated Data (AEAD) modes are designed to provide confidentiality, integrity, and authenticity simultaneously. AES-GCM is a widely adopted AEAD mode, standardized by NIST (SP 800-38D) [19].

GCM combines AES in Counter (CTR) mode for encryption with a universal hash function called GHASH for authentication.

Inputs:

- AES Key (K): 128, 192, or 256 bits.
- Nonce (N): Initialization Vector (IV). Must be unique for every encryption with the same key. Typically 96 bits (12 bytes) for optimal performance, but other lengths are allowed. Nonce uniqueness is critical for GCM security.
- Plaintext (P): The message to be encrypted.
- Associated Data (A): Optional data that needs integrity and authenticity protection but does not need to be encrypted (e.g., packet headers).

Outputs:

- Ciphertext (C): Same length as Plaintext.
- Authentication Tag (T): A fixed-length MAC (e.g., 128 bits/16 bytes).

Process Overview:

1. Key Derivation: An internal hash key H is derived by encrypting the all-zero block with AES: $H = \text{AES-Encrypt}(K, 0^{128})$.
2. Nonce Processing: A unique counter block J_0 is derived from the nonce N. If $\text{len}(N) = 96$, then $J_0 = N||0^{31}||1$. Otherwise, GHASH is used to process N.
3. Encryption (CTR Mode): Subsequent counter blocks $(J_1, J_2, \ldots)$ are generated by incrementing J_0. These counter blocks are encrypted using AES with key K: $\text{KeystreamBlock}_i = \text{AES-Encrypt}(K, J_i)$. The plaintext P is XORed with the generated keystream to produce the ciphertext C.
4. Authentication (GHASH): The GHASH function computes a MAC over the Associated Data A and the Ciphertext C. GHASH operates in the finite field $\mathbb{GF}(2^{128})$ defined by a specific irreducible polynomial. It involves multiplying blocks of A and C (padded appropriately) by powers of the hash key H and XORing the results. The length of A and C is also included in the hash.
5. Tag Generation: The final Authentication Tag T is generated by encrypting the first counter block J_0 and XORing the result with the output of GHASH: $T = \text{GHASH}(H, A, C) \oplus \text{AES-Encrypt}(K, J_0)$.

Decryption and Verification: The recipient performs the inverse process. They use CTR mode with the received nonce to decrypt the ciphertext. They independently compute the expected authentication tag using the received nonce, associated data, and the decrypted plaintext (or alternatively, the received ciphertext). If the computed tag matches the received tag T, the decryption is considered successful, and the data is deemed authentic and unmodified. If the tags do not match, the decryption fails, indicating potential tampering or corruption.

Importance of Nonce Uniqueness: If a nonce is ever reused with the same key for two different messages, GCM security collapses catastrophically. It leaks information about the plaintexts and allows an attacker to forge authentication tags. Therefore, ensuring nonce uniqueness (e.g., using counters or random generation with sufficient length) is paramount when using AES-GCM. In this thesis, nonces are derived using HKDF (Section 2.2.3) to ensure uniqueness per session.

2.2.3 Key Derivation Functions (KDF)

Cryptographic keys used directly in algorithms like AES often need to possess specific properties, such as uniform randomness and fixed length. However, shared secrets established through protocols like ECDH might not inherently have these properties (e.g., the x-coordinate of the ECDH point). Key Derivation Functions (KDFs) are used to transform initial, potentially less ideal keying material (like an ECDH shared secret) into one or more cryptographically strong keys suitable for specific algorithms.

HMAC-based KDF (HKDF)

HKDF, specified in RFC 5869 [20], is a popular and robust KDF based on the HMAC (Hash-based Message Authentication Code) construction. It follows an "extract-then-expand" paradigm, ensuring that the resulting keys are cryptographically strong even if the input keying material has some bias or is not uniformly random. HKDF requires an underlying hash function (e.g., SHA-256).

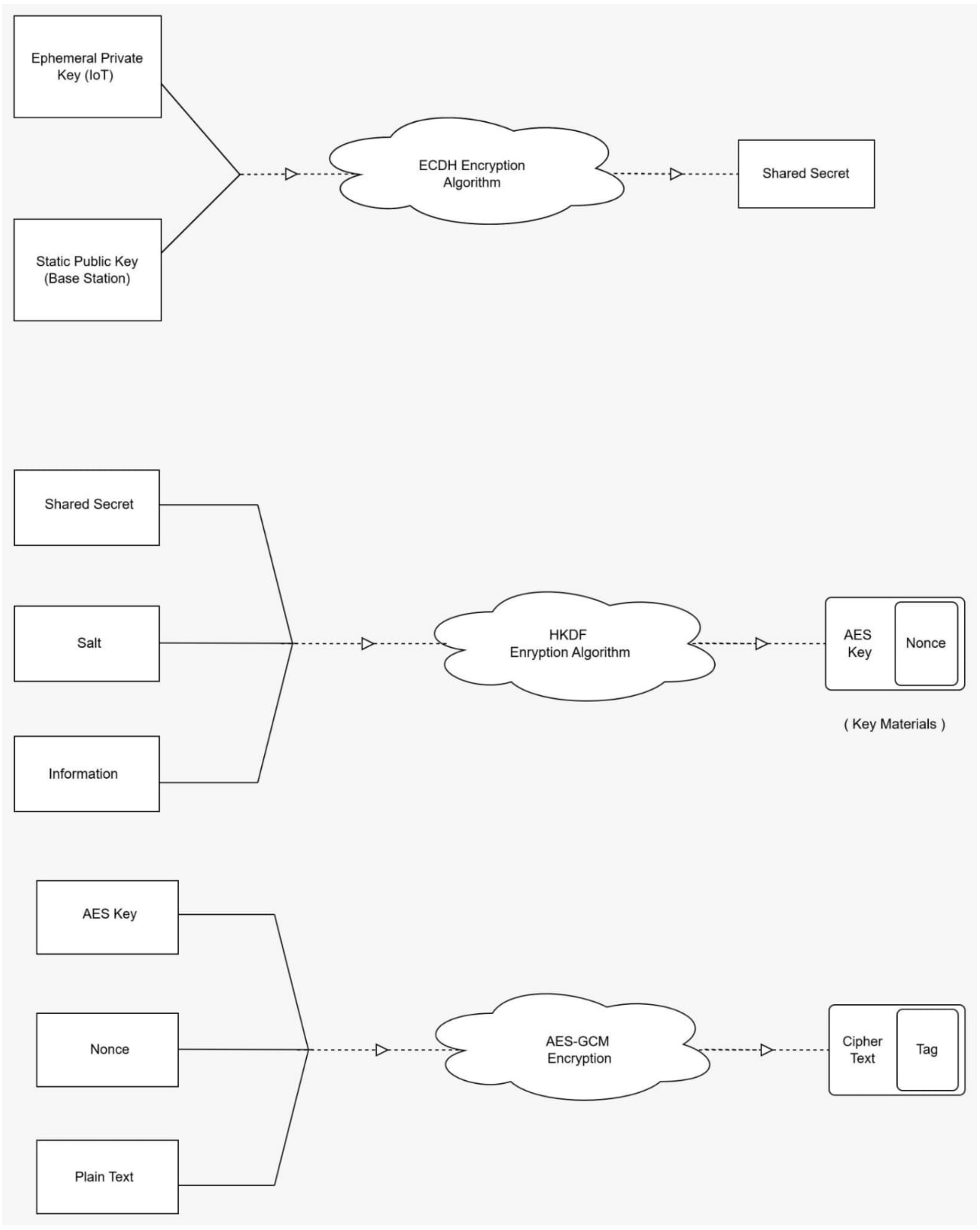

Figure 2.4: AES-GCM Encryption Process

HMAC Reminder: $\text{HMAC-Hash}(\text{key}, \text{message}) = \text{Hash}((\text{key}\oplus\text{opad})||\text{Hash}((\text{key}\oplus\text{ipad})||\text{message}))$ where ipad and opad are fixed padding constants.

HKDF Phases:

- **Extract Phase:** This phase takes the potentially non-uniform Input Keying Material (IKM, e.g., the ECDH shared secret x_s) and an optional salt value, and outputs a fixed-length Pseudorandom Key (PRK). The salt is ideally a random, non-secret value; if not provided, it defaults to a string of zeros. Using a salt enhances security by ensuring uniqueness even if the same IKM is used in different contexts.

  $$\text{PRK} = \text{HMAC-Hash}(\text{salt}, \text{IKM}) \tag{2.2}$$

  The output PRK has the length of the hash function's output (e.g., 32 bytes for SHA-256).

- **Expand Phase:** This phase takes the PRK generated by the extract phase, an optional context-specific string `info`, and the desired length L of the Output Keying Material (OKM). It generates OKM of length L bytes. The `info` string allows binding the derived keys to the specific application protocol or context, preventing potential cross-protocol attacks where keys derived for one purpose might be misused in another. The expansion generates keying material iteratively in blocks:

  $$T(0) = \text{"" (empty string)} \tag{2.3}$$
  $$T(1) = \text{HMAC-Hash}(\text{PRK}, T(0)|\text{info}|0x01) \tag{2.4}$$
  $$T(2) = \text{HMAC-Hash}(\text{PRK}, T(1)|\text{info}|0x02) \tag{2.5}$$
  $$\ldots \tag{2.6}$$
  $$T(N) = \text{HMAC-Hash}(\text{PRK}, T(N-1)|\text{info}|N) \tag{2.7}$$

  where $N = \lceil L/\text{HashLen} \rceil$, HashLen is the output length of the hash function in bytes, $|$ denotes concatenation, and the final byte is a counter. The OKM is the first L bytes of the concatenated sequence $T(1)|T(2)|\ldots|T(N)$.

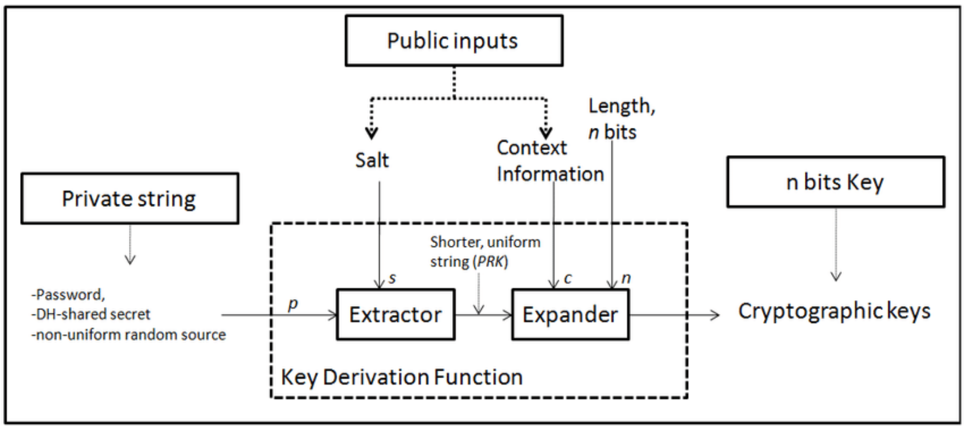

Figure 2.5: HKDF Extract-then-Expand Process

In this thesis, HKDF (with SHA-256) is used to derive the 128-bit AES key and the 96-bit AES-GCM nonce from the ECDH shared secret (x_s).

Role of Salt and Info Parameters

- `salt`: The primary role of the salt in the Extract phase is to "de-bias" the input keying material and ensure independence between different HKDF applications potentially using the same IKM. Even if the IKM is not perfectly random or is reused, using different salts will result in different PRK values. While optional, providing a random (though not necessarily secret) salt is highly recommended for robustness. In this project, distinct salts like "`uplink_salt`" and "`downlink_salt`" could be used.
- `info`: The info parameter in the Expand phase provides context separation. It cryptographically binds the derived OKM to the specific context in which it will be used. For example, using `info` = "`aes-key`" to derive the AES key and `info` = "`aes-nonce`" to derive the nonce ensures that these two outputs are distinct and cannot be mistaken for each other or for keys derived for a different purpose (e.g., `info` = "`mac-key`"). This is crucial for preventing subtle cryptographic attacks that exploit key reuse across different contexts or algorithms.

By employing HKDF with appropriate salt and info values, the system ensures that the symmetric keys derived from the ECDH shared secret possess strong cryptographic properties and are securely bound to their intended use within the hybrid encryption protocol.

2.3 Blockchain and Smart Contract Fundamentals

The second core component of the proposed system leverages blockchain technology, specifically smart contracts, to implement the trustless incentive mechanism. This section provides the necessary background on Distributed Ledger Technology (DLT), the Ethereum platform, and the principles of smart contracts.

2.3.1 Distributed Ledger Technology (DLT) Concepts

Distributed Ledger Technology (DLT) refers to a broad category of systems that record transactions and their details across multiple locations simultaneously, without a central data store or administrator. A blockchain is a specific type of DLT where records (transactions) are grouped into blocks, and each block is cryptographically linked to the previous one using a hash, forming a chronological chain [21]. Key characteristics of DLTs, particularly public blockchains like Ethereum, include:

- **Decentralization:** The ledger is maintained by a network of nodes (computers) rather than a single central authority. No single entity has complete control over the ledger or its history.
- **Transparency:** In public blockchains, while user identities might be pseudonymous (represented by addresses), the transactions themselves are typically visible to anyone participating in or observing the network.
- **Immutability:** Once a block of transactions is validated by the network through a consensus mechanism and added to the chain, it becomes extremely difficult and computationally expensive to alter or delete it. This resistance to tampering provides a high degree of data integrity for the recorded history.

- **Consensus Mechanisms:** Distributed networks require mechanisms to agree on the validity of transactions and the current state of the ledger. Common mechanisms include Proof-of-Work (PoW), used historically by Bitcoin and Ethereum, and Proof-of-Stake (PoS), which Ethereum has transitioned to. These mechanisms ensure that all participating nodes converge on a single, consistent version of the ledger history [24].

These properties collectively enable the creation of trustless systems where participants can interact and transact with a high degree of confidence in the integrity of the record, even without knowing or trusting each other directly.

2.3.2 Ethereum Blockchain Platform

While Bitcoin introduced the concept of a peer-to-peer electronic cash system using blockchain, Ethereum, proposed by Vitalik Buterin [22], extended the technology's capabilities significantly by introducing the concept of smart contracts. Ethereum is a decentralized, open-source blockchain platform designed to be a global computer for executing decentralized applications (dApps).

Ethereum Virtual Machine (EVM)

At the heart of Ethereum lies the Ethereum Virtual Machine (EVM). The EVM is a quasi-Turing complete, sandboxed virtual machine embedded within each Ethereum node [23]. Its purpose is to execute arbitrary code, known as smart contracts, specified in transactions. Every node in the Ethereum network runs the EVM as part of the block verification protocol, executing the same instructions based on the transactions included in a block. This replicated state machine execution ensures that all nodes reach the same resulting state after processing a block, maintaining consensus across the network. Smart contract code is typically written in high-level languages (like Solidity) and then compiled down to EVM bytecode for execution.

Externally Owned Accounts (EOAs) vs. Contract Accounts

Ethereum features two main types of accounts:

- **Externally Owned Accounts (EOAs):** These accounts are controlled by users via private keys. An EOA can initiate transactions (sending Ether or triggering contract code execution) and sign them using its corresponding private key. EOAs do not have code associated with them. Your typical user wallet manages an EOA.
- **Contract Accounts:** These accounts are controlled by the code deployed to them (the smart contract code). They have an associated address, just like EOAs, but they also contain compiled EVM bytecode. A contract account cannot initiate transactions on its own; its code is executed only when it receives a transaction from an EOA or a message call from another contract. The smart contract developed in this thesis resides in a contract account.

Gas and Transaction Costs

Performing operations on the Ethereum blockchain (e.g., sending Ether, executing smart contract functions, storing data) consumes computational resources across the network. To prevent deliberate or accidental resource exhaustion and to compensate the nodes (validators in PoS) performing the computation and validation, Ethereum employs a mechanism called Gas [25].

- **Gas Units:** Every low-level operation executable by the EVM (e.g., ADD, STORE, KECCAK256) has a fixed cost specified in units of Gas. More complex operations consume more Gas.
- **Gas Price:** Users initiating transactions specify a Gas Price they are willing to pay per unit of Gas, typically denominated in Gwei (1 Gwei = 10^{-9} Ether).
- **Gas Limit:** Users also set a Gas Limit, the maximum amount of Gas they are willing to spend on a transaction.
- **Transaction Fee:** The total transaction fee is calculated as Gas Used * Gas Price. This fee is paid by the transaction initiator (from their EOA) in Ether.

If a transaction runs out of Gas before completing execution (i.e., Gas Used exceeds Gas Limit), the execution halts, any state changes are reverted, but the transaction fee for the Gas consumed up to that point is still charged. This mechanism ensures that network resources are paid for and incentivizes efficient smart contract code development. The concept of Gas cost is a critical factor in evaluating the economic viability of blockchain-based solutions, including the reward mechanism proposed in this thesis.

2.3.3 Smart Contracts: Principles and Lifecycle

A smart contract, in the context of Ethereum, is essentially a program that runs on the blockchain at a specific address [26]. It automatically executes predefined actions when specific conditions, encoded within its logic, are met.

Principles:

- **Code is Law:** Once deployed to the blockchain, the smart contract's code typically cannot be altered (though upgrade patterns exist). Its execution is deterministic and enforced by the network consensus rules.
- **Autonomy:** Smart contracts can operate autonomously without human intervention once deployed, reacting to incoming transactions or calls according to their programmed logic.
- **Trustless Execution:** Because execution is handled by the decentralized network and governed by consensus, users can interact with a smart contract with confidence that it will execute exactly as written, without needing to trust the contract creator or any single intermediary.
- **State Persistence:** Smart contracts can hold state (data stored in variables) persistently on the blockchain ledger.

Lifecycle:

- **Development:** Contracts are typically written in a high-level programming language. Solidity is the most prominent, statically-typed language designed specifically for writing smart contracts targeting the EVM [27]. It provides constructs for defining state variables, functions (with visibility controls like public, private, internal, external), events (for logging), modifiers (for reusable checks), and interacting with other contracts. Developers use frameworks like Hardhat or Truffle to write, compile, and test their Solidity code.

- **Compilation:** The high-level code (e.g., Solidity) is compiled into EVM bytecode.
- Deployment: The compiled bytecode is sent to the blockchain via a special transaction initiated from an EOA. This transaction creates a new contract account on the blockchain, associates the bytecode with that account's address, and runs any constructor logic defined in the contract. Deployment incurs a Gas cost.
- **Interaction:** Once deployed, EOAs or other contracts can interact with the smart contract by sending transactions that call its public functions. These interactions read or modify the contract's state and may involve transferring Ether. Interactions also consume Gas.
- **(Potential) Termination/Upgrade:** Basic smart contracts are immutable. However, various design patterns (e.g., proxy patterns, data separation) can be implemented to allow for upgrading contract logic or effectively terminating a contract's usefulness (e.g., by transferring ownership or pausing functionality), though the original code remains on the blockchain.

The smart contract designed in this thesis (`MessageReward`) utilizes Solidity to define the logic for storing reward parameters, verifying the cryptographic proof provided by intermediaries, and disbursing rewards, thereby embodying the principles of automated, trustless execution central to this research.

2.4 Related Work

This section reviews existing literature and technologies relevant to the core challenges addressed in this thesis: secure communication for constrained IoT devices and incentivizing network participation. It examines prior work in secure IoT protocols, the application of blockchain in IoT contexts, and cryptoeconomic incentive models, culminating in an analysis that identifies the specific research gap this thesis aims to fill.

2.4.1 Existing Secure Communication Protocols for IoT

Securing communication in resource-constrained IoT environments remains an active area of research and standardization. Several approaches exist, each with its own trade-offs:

- **Optimized TLS/DTLS:** As discussed in Section 2.1.2, standard TLS/DTLS often imposes prohibitive overhead. Efforts to mitigate this include:
    - **TLS/DTLS Pre-Shared Keys (PSK):** Cipher suites based on PSK avoid the computationally expensive public-key operations of the standard handshake, relying instead on symmetric keys pre-shared between endpoints [28]. However, scalable and secure distribution and management of unique PSKs for numerous devices remain significant challenges.
    - **TLS/DTLS Raw Public Keys (RPK):** These modes replace X.509 certificates with bare public keys, reducing handshake size and eliminating certificate validation complexity [29]. While reducing overhead, they still rely on public-key cryptog-

raphy for the key exchange and require mechanisms to securely bind keys to identities.

  ◦ **Session Resumption:** Techniques like session tickets or session IDs allow endpoints to reuse cryptographic parameters from a previous session, significantly shortening subsequent handshakes [30]. This is effective for devices communicating frequently with the same server but doesn't help with the initial full handshake cost.

  ◦ **DTLS Connection ID (CID):** Introduced in DTLS 1.2 and mandatory in 1.3, CID helps maintain session context across IP address/port changes (common in NAT or mobile scenarios) without a full handshake, improving efficiency for certain network conditions [31].

- **Application-Layer Security Protocols:** Protocols like OSCORE (Object Security for Constrained RESTful Environments) provide end-to-end security at the application layer, specifically designed for CoAP (Constrained Application Protocol) [32]. OSCORE protects CoAP messages, including headers and payload, between endpoints, potentially traversing unsecured intermediaries. It offers granular security but is tightly coupled with CoAP and requires its own key management infrastructure. Similar application-layer security can be implemented over protocols like MQTT, but often relies on TLS for transport security to the broker, leaving end-to-end security dependent on application-level implementation.

- **LPWAN Security Mechanisms (e.g., LoRaWAN):** Low-Power Wide-Area Networks like LoRaWAN incorporate security features at the MAC/Network layers [33]. LoRaWAN uses unique device identifiers (DevEUI, JoinEUI) and pre-shared keys (AppKey) to derive session keys (AppSKey for application payload encryption/integrity, NwkSKey for MAC command integrity) during an over-the-air activation (OTAA) join procedure. While providing link-layer security and some level of end-device authentication, the session keys are typically known to the Network Server and potentially the Join Server, meaning data is not strictly end-to-end encrypted between the device and the final application owner if these network elements are untrusted. Furthermore, the scope of integrity protection and replay prevention has evolved across LoRaWAN versions.

While these solutions offer valuable security properties, they may not fully address the specific combination of requirements tackled in this thesis: strong end-to-end security (including Perfect Forward Secrecy often lacking in PSK modes or basic LoRaWAN), resilience against fully untrusted intermediaries, minimal overhead for highly constrained devices, and seamless integration with a verifiable, per-message incentive layer. The hybrid ECDH/AES-GCM protocol proposed here aims to provide robust, session-independent end-to-end security with PFS, specifically designed to operate independently of intermediary trust.

2.4.2 Blockchain Applications in IoT

Blockchain technology's properties – decentralization, immutability, transparency – have attracted significant interest for addressing various challenges within the IoT domain beyond just payment or incentives:

- **Data Integrity and Auditability:** Storing hashes of IoT sensor data or critical event logs on a blockchain creates a tamper-evident audit trail. Any modification to the off-chain data would result in a hash mismatch, detectable by querying the blockchain [34].

- **Decentralized Access Control:** Smart contracts can be used to define and enforce complex access control policies for IoT devices or data streams. Users or other devices could request access via the blockchain, with the smart contract granting or denying permission based on predefined rules and potentially recorded credentials or tokens [35].
- **Supply Chain Management and Provenance:** Combining IoT sensors (tracking location, temperature, etc.) with blockchain allows for creating immutable records of a product's journey through the supply chain, enhancing transparency and verifying authenticity [36].
- **Device Identity and Management:** Blockchain can provide a decentralized registry for IoT device identities, potentially facilitating secure onboarding, firmware updates, and decommissioning [37].

While these applications demonstrate the versatility of blockchain in IoT, they generally focus on using the ledger as a secure database or access control manager. This thesis utilizes blockchain differently – not primarily for storing IoT data itself, but as the core infrastructure for a trustless execution environment (the smart contract) that verifies cryptographic proof of an action (data relay) and automates the resulting incentive.

2.4.3 Incentive Mechanisms for Network Participation (Cryptoeconomics)

Motivating disparate, independently owned nodes to contribute resources (bandwidth, storage, computation, connectivity) to a distributed network is a fundamental challenge. Cryptoeconomics, the intersection of cryptography, economics, and game theory, provides tools to design systems where participants are economically incentivized, often via native digital tokens, to behave honestly and contribute positively to the network's function.

Early examples explored incentivizing file storage (e.g., Filecoin, Storj [39]) or bandwidth sharing. More relevant to this thesis is the incentivization of network connectivity and data relay.

- **The Helium Network:** Helium represents a prominent, large-scale attempt to build decentralized wireless networks (initially LoRaWAN, now also 5G) using cryptoeconomic incentives [40].
  - **Model:** Individuals or companies deploy "Hotspots" (gateways). These Hotspots provide wireless coverage and relay data packets from IoT devices to the internet (Network Servers).
  - **Incentives:** Hotspot owners earn Helium's native tokens (originally HNT, now specific tokens like IOT for the LoRaWAN network) for contributing to the network. Rewards are primarily based on: Proof-of-Coverage (PoC) and Data Transfer.
  - **Blockchain Role:** The Helium blockchain is used to record Hotspot identities, validate PoC challenges, track data transfer credits, and mint/distribute token rewards.
  - **Comparison with Thesis Project:** Similarities exist in using blockchain for incentives. Key differences lie in scale/goal (Helium: global infrastructure; Thesis: specific secure interaction), proof mechanism (Helium: PoC/Network validation; Thesis: receiver crypto proof), trust focus (Helium relies more on network elements;

Thesis assumes untrusted intermediary for content), and granularity (Helium: aggregated rewards; Thesis: per-message reward).

- **Other Models:** Various academic proposals and smaller projects have explored reputation systems, auctions, or simpler smart contract-based payments for relay or computation tasks in different network contexts [41].

These existing models provide valuable insights. However, the mechanism proposed in this thesis differentiates itself by its direct, cryptographically verifiable proof linked to the successful reception of an end-to-end secured message.

Table 2.2: Comparison of Related Works/Protocols

Feature/Protocol	Optimized TLS/DTLS	OSCORE	LoRaWAN Security	Helium Network	Proposed System
Security Goal	Link/E2E (varies)	E2E (CoAP)	Link/Network	Link (LoRaWAN)	E2E
PFS Support	Yes (ECDHE) / No (PSK)	Yes (with ECDH)	Limited (Join only)	N/A (Link Layer)	Yes (Ephemeral ECDH)
Incentive Type	None	None	None	Macro/Network	Micro/Per-message
Proof of Relay	None	None	None	PoC/Network Agg.	Receiver Crypto Proof
Intermediary Trust	Varies	Untrusted (Content)	Trusted (Network Server)	Implicit (Hotspot)	Untrusted (Content)
Target Env.	General Web/IoT	Constrained IoT (CoAP)	Constrained IoT (LPWAN)	Large-Scale Infra	Constrained IoT

2.4.4 Analysis of Existing Solutions and Identification of Research Gaps

The review highlights several points:

- **Security Protocol Gaps:** Existing protocols have trade-offs regarding PFS, intermediary trust assumptions, or protocol specificity. The proposed hybrid protocol targets robust E2E security with PFS over untrusted intermediaries.
- **Blockchain Application Focus:** Most IoT blockchain uses focus on data integrity or access control, differing from this thesis's use of a smart contract as a trustless execution engine for verifying relay proof and automating micro-incentives.
- **Incentive Model Comparison:** While models like Helium incentivize infrastructure, the proposed system offers a fine-grained, per-message incentive based on direct, receiver-generated cryptographic proof, suitable for scenarios demanding high reliability for individual messages under stricter intermediary trust assumptions.

Based on this analysis, the primary research gap addressed by this thesis remains the lack of an integrated system that cohesively combines:

- A lightweight hybrid cryptographic protocol providing robust E2E security guarantees (Confidentiality, Integrity, Authenticity, PFS) between resource-constrained endpoints, operating securely over fully untrusted intermediaries.
- A simple, trustless, and verifiable micro-incentive mechanism via a smart contract, rewarding intermediaries per message based on cryptographic proof generated and verified solely by the endpoints.

This thesis aims to fill this gap by designing, implementing, and evaluating such an integrated system.

Chapter 3

System Design and Architecture

This chapter presents the detailed design and architecture of the proposed system for secure, incentivized IoT communication. It outlines the overall system structure, defines the roles and responsibilities of each component, specifies the trust assumptions and threat model, and provides a step-by-step specification of the hybrid encryption protocol used for both uplink and downlink communication. Finally, it details the design of the smart contract responsible for the trustless reward mechanism.

3.1 Overall System Architecture

The system comprises five main components interacting across two distinct networks: the data communication network and the blockchain network.

3.1.1 Component Roles and Responsibilities

- IoT End Device: Source/recipient of data. Resource-constrained. Holds static keys (sk_I, pk_I). Generates ephemeral keys. Performs ECDH, AES-GCM encryption/decryption, ECDSA signing/verification. Constructs/parses packets. Generates downlink reward proof.
- Intermediary Node: Relays encrypted packets. Untrusted re: content. Has blockchain EOA. Receives reward proof. Sends `claimReward` transaction to smart contract.
- Base Station: Counterpart to IoT device(s). Less constrained. Holds static keys (sk_B, pk_B). Performs ECDH, AES-GCM encryption/decryption, ECDSA signing/verification. Constructs/parses packets. Generates uplink reward proof (keccak256 + ECDSA). Provides proof to Intermediary. Deploys/funds contract.
- Blockchain Network (EVM-compatible): Decentralized execution environment. Executes EVM code. Validates transactions. Maintains ledger state. Ensures immutability.
- MessageReward Smart Contract: Automated reward arbiter on blockchain. Stores receiver public key(s), reward amount. Holds funds. Provides `claimReward(hash, signature)` function. Verifies signature, checks claim status, checks funds. Transfers reward to `msg.sender`. Emits events.

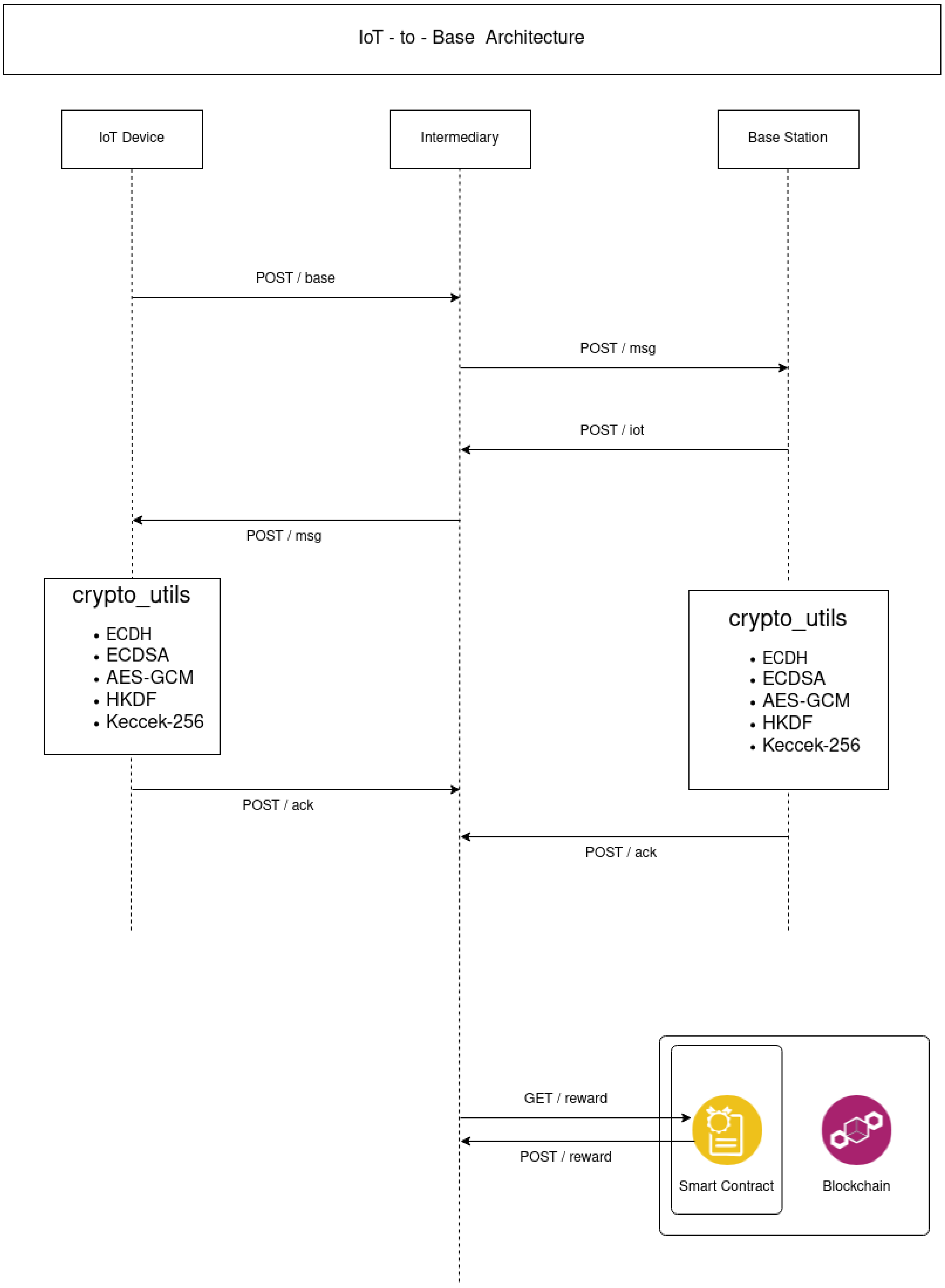

Figure 3.1: Detailed System Architecture

3.1.2 Trust Assumptions and Threat Model

- **Trust Assumptions:**
  - Endpoints (IoT, Base Station): Honest, correctly implemented. Static private keys secure. Static public keys authentic (pre-shared).
  - Intermediary Node: Untrusted re: message content (may eavesdrop, modify, drop, replay, inject). Rational (motivated by reward). Trusted to manage own EOA.
  - Blockchain Network: Secure per standard model (e.g., 51% attack resistant). Consensus ensures correct execution.
  - Smart Contract Code: Deployed bytecode correctly implements logic (functionally correct).
- **Threat Model:**
  - Attacker Capabilities (M controls network): Eavesdrop, Modify, Inject, Replay, Drop packets. MitM (Intermediary).
  - Attacker Goals: Violate confidentiality, integrity, authenticity. Replay attacks. Fraudulent reward claims.
  - Out of Scope: Physical attacks, endpoint compromise (key extraction), side-channels, DoS vs endpoints/blockchain, consensus attacks, advanced contract vulnerabilities.

3.2 Hybrid Encryption Protocol Design

Combines ECC (ECDH, ECDSA) with AES-GCM.

3.2.1 Key Management Strategy

Static Key Pairs (Pre-distribution/Configuration)

- IoT Device (I): Unique static ECC pair (sk_I, pk_I) (NIST P-256).
- Base Station (B): Static ECC pair (sk_B, pk_B) (NIST P-256).
- Requirement: I must know authentic pk_B, B must know authentic pk_I. Secure pre-distribution assumed (out of scope). Used for ECDSA.

Ephemeral Key Pair Generation and Lifecycle

- Sender generates fresh ephemeral ECC pair (esk, epk) per session/message.
- esk used once for ECDH, then discarded securely (MUST NOT be reused).
- epk transmitted plaintext in message header.
- Ensures Perfect Forward Secrecy (PFS). Compromise of static keys doesn't compromise past sessions.

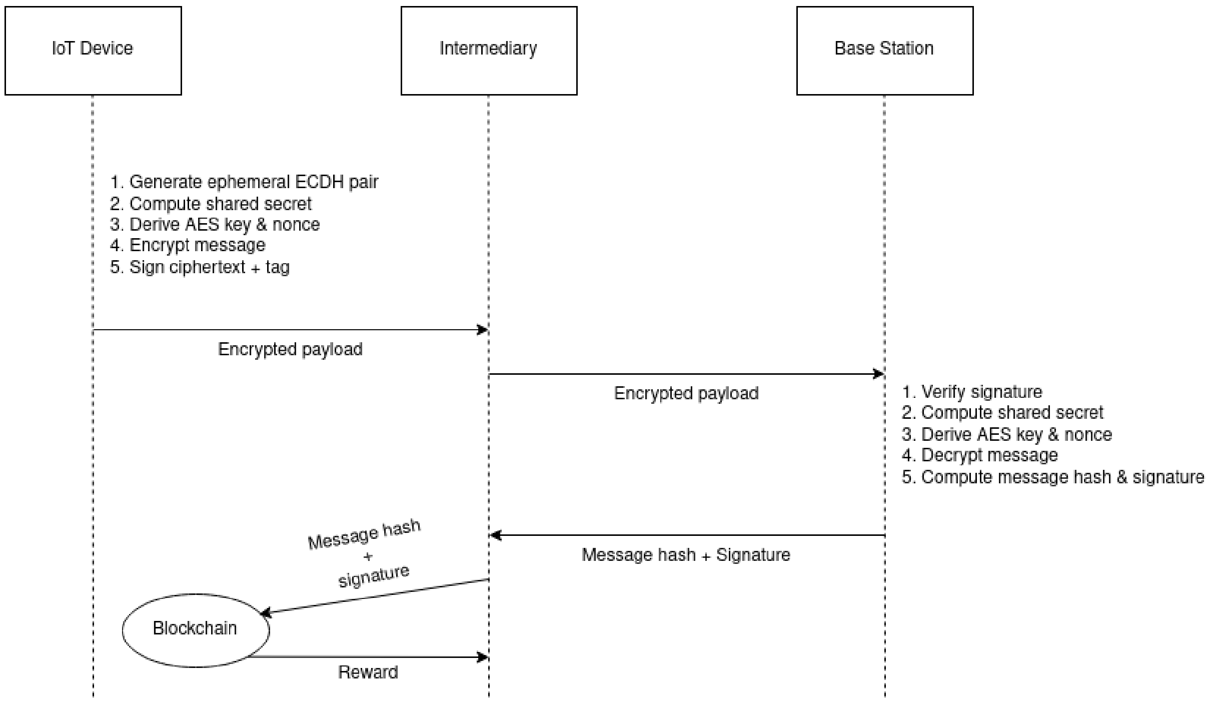

Figure 3.2: Uplink Sequence Diagram

3.2.2 Uplink Communication Protocol (IoT -> Base Station)

Step-by-Step Procedure (Uplink)

1. **IoT Device (Sender):**
   (a) Get payload.
   (b) Generate ephemeral pair (esk_I, epk_I).
   (c) Compute ECDH secret: $S = \text{Point}(\text{esk\_I} \times \text{pk\_B})$. Use x-coord S_x.
   (d) Derive keys: $(\text{aes_key}, \text{nonce}) = $ HKDF(salt="uplink", IKM=S_x, info="session_keys", L=32).
   (e) Encrypt: $(\text{ciphertext}, \text{auth_tag}) = $ AES-GCM-Encrypt(key=aes_key, nonce=nonce, plaintext=payl
   (f) Prepare data for signing: $\text{data\_to\_sign} = \text{epk\_I}||\text{ciphertext}||\text{auth\_tag}$.
   (g) Compute hash: $\text{hash\_to\_sign} = \text{SHA256}(\text{data\_to\_sign})$.
   (h) Sign hash: $\text{sig\_I} = \text{ECDSA\_Sign}(\text{sk\_I}, \text{hash\_to\_sign})$.
   (i) Construct packet: $(\text{epk\_I}, \text{ciphertext}, \text{auth\_tag}, \text{sig\_I})$.
   (j) Send packet to Intermediary.
   (k) Discard esk_I.
2. **Intermediary:**
   (a) Receive packet.
   (b) Forward unmodified to Base Station.
3. **Base Station (Receiver):**

(a) Receive packet. Parse components.

(b) Reconstruct data: data_to_sign' $=$ epk_I$||$ciphertext$||$auth_tag.

(c) Compute hash: hash_to_sign' $=$ SHA256(data_to_sign').

(d) Verify signature: is_valid_sig $=$ ECDSA_Verify(pk_I, hash_to_sign', sig_I). If invalid, reject.

(e) Compute ECDH secret: $S' =$ Point(sk_B $\times$ epk_I). Use S'_x.

(f) Derive keys: (aes_key', nonce') $=$ HKDF(salt="uplink", IKM=S'_x, info="session_keys", L=32).

(g) Decrypt/verify payload: payload' $=$ AES-GCM-Decrypt($\ldots$).

(h) If decryption fails (tag mismatch), reject.

(i) If successful, process payload'.

(j) Generate Reward Proof: message_hash $=$ keccak256(raw_received_packet_bytes). reward_sig $=$ ECDSA_Sign(sk_B, message_hash).

(k) Provide proof (message_hash, reward_sig) to Intermediary.

Data Packet Structure (Uplink)

Table 3.1: Uplink Data Packet Structure

Field	Length (Bytes)	Description
Header	(Variable)	Optional: Sender ID, Receiver ID, Type, Lengths
Ephemeral PubKey	65	Sender's (IoT) epk_I, uncompressed (0x04 + x + y)
Ciphertext	Variable	AES-GCM encrypted payload
Authentication Tag	16	AES-GCM tag (auth_tag) covering Ciphertext and AAD (epk_I)
Signature	Variable ($\sim$71)	ECDSA signature (sig_I) over SHA256(epk_I || ciphertext || tag)
Total (approx)		$\sim$152 + Payload Size + Header Size

Note: Compressed public keys ($\sim$33 bytes) could save space. Signature length varies (e.g., raw vs DER).

3.2.3 Downlink Communication Protocol (Base Station -> IoT)

Symmetric to uplink.

Step-by-Step Procedure (Downlink)

1. **Base Station (Sender):**

(a) Get payload.

(b) Generate ephemeral pair (esk_B, epk_B).

(c) Compute ECDH secret: $S =$ Point(esk_B $\times$ pk_I). Use S_x.

(d) Derive keys: (aes_key, nonce) $=$ HKDF(salt="downlink", IKM=$S_x, \ldots$).

(e) Encrypt: (ciphertext, auth_tag) $=$ AES-GCM-Encrypt($\ldots$, aad=epk_B).

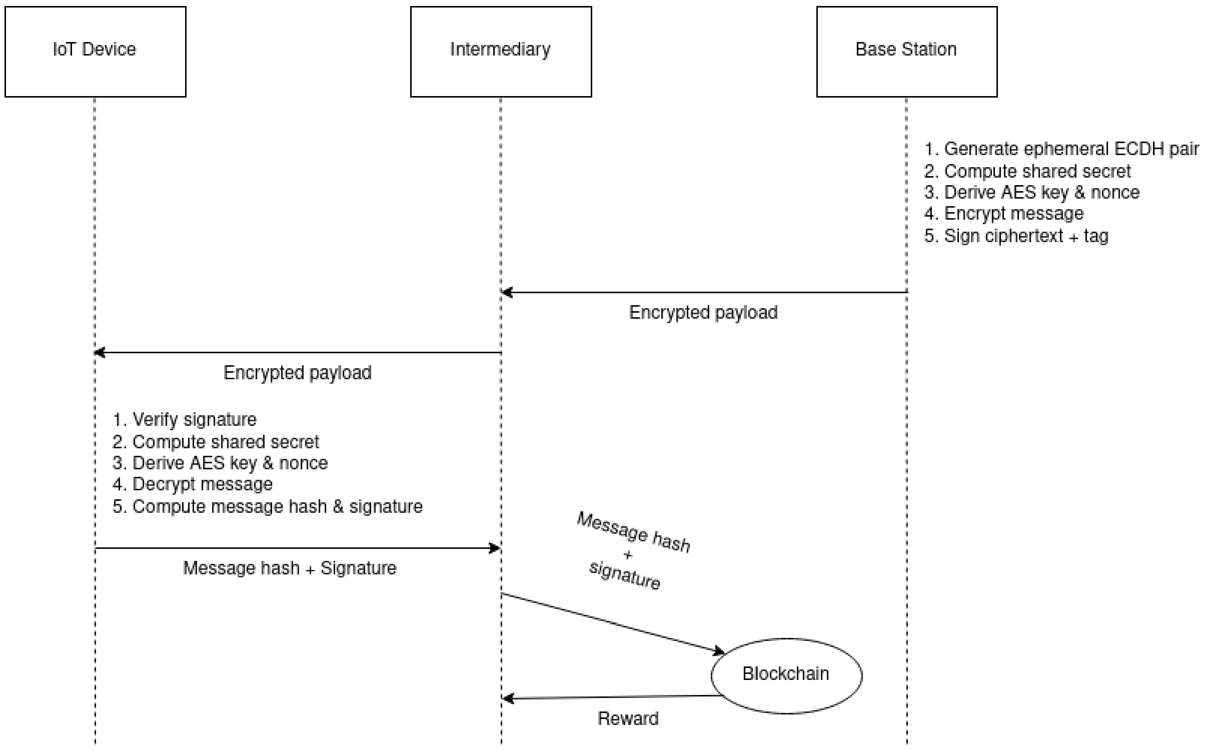

Figure 3.3: Downlink Sequence Diagram

(f) Prepare/hash: data_to_sign $=$ epk_B$||\dots$, hash_to_sign $=$ SHA256$(\dots)$.

(g) Sign hash: sig_B $=$ ECDSA_Sign(sk_B, hash_to_sign).

(h) Construct packet: (epk_B, ciphertext, auth_tag, sig_B).

(i) Send packet to Intermediary.

(j) Discard esk_B.

2. **Intermediary:**

   (a) Receive packet.

   (b) Forward unmodified to IoT Device.

3. **IoT Device (Receiver):**

   (a) Receive packet. Parse components.

   (b) Reconstruct/hash: data_to_sign’ $=$ epk_B$||\dots$, hash_to_sign’ $=$ SHA256$(\dots)$.

   (c) Verify signature: is_valid_sig $=$ ECDSA_Verify(pk_B, hash_to_sign’, sig_B). If invalid, reject.

   (d) Compute ECDH secret: $S' =$ Point(sk_I $\times$ epk_B). Use S'_x.

   (e) Derive keys: (aes_key’, nonce’) $=$ HKDF(salt="downlink", IKM=$S'_x, \dots$).

   (f) Decrypt/verify payload: payload’ $=$ AES-GCM-Decrypt$(\dots,$ aad=epk_B$)$.

   (g) If decryption fails, reject.

   (h) If successful, process payload’.

(i) Generate Reward Proof: message_hash = keccak256(raw_received_packet_bytes). reward_sig = ECDSA_Sign(sk_I, message_hash).

(j) Provide proof (message_hash, reward_sig) to Intermediary.

Data Packet Structure (Downlink)

Identical structure to uplink (Table 3.1), roles reversed (Sender=Base Station, Receiver=IoT).

3.3 Blockchain-Based Reward Mechanism

Details the trustless reward mechanism using an Ethereum smart contract.

3.3.1 MessageReward Smart Contract Design

Decentralized escrow and automated verification engine on the blockchain.

State Variables

- `address public immutable baseStationAddress;`: Address derived from pk_B (signer for uplink proof).
- `address public immutable iotDeviceAddress;`: Address derived from pk_I (signer for downlink proof).
- `uint256 public immutable fixedRewardAmount;`: Reward amount per message in Wei (10^{18} Wei = 1 Ether).
- `mapping(bytes32 => bool) public claimedHashes;`: Tracks claimed message hashes (key: keccak256 hash) to prevent double-spending.
- `(Optional) address public owner;`: For admin functions if needed (e.g., using OpenZeppelin's Ownable).

Constructor Logic and Parameters

Executed once at deployment to initialize immutable parameters.

- Parameters: `_initialBaseStation` (address), `_initialIotDevice` (address), `_initialRewardWei` (uint256).
- Logic:
  - Validate inputs (`require` addresses are not zero, reward > 0).
  - Initialize state variables: `baseStationAddress`, `iotDeviceAddress`, `fixedRewardAm`
  - Emit `ConfigUpdated` event with initial values for transparency.

Illustrative Solidity Snippet (Constructor):

Listing 3.1: Illustrative Solidity Constructor

```
contract MessageReward {
    // ... (Imports, state variables, events as described) ...
    constructor(address _initialBaseStation, address
       _initialIotDevice, uint256 _initialRewardWei) {
        require(_initialBaseStation != address(0), "Invalid
           Base Station address");
        require(_initialIotDevice != address(0), "Invalid IoT
           Device address");
        require(_initialRewardWei > 0, "Reward must be
           positive"); // Added check
        baseStationAddress = _initialBaseStation;
        iotDeviceAddress = _initialIotDevice;
        fixedRewardAmount = _initialRewardWei;
        emit ConfigUpdated(_initialBaseStation,
           _initialIotDevice, _initialRewardWei);
    }

    // ... (claimReward function and receive function) ...
}
```

claimReward Function Detailed Logic

Primary function called by Intermediary.

- Signature: `function claimReward(bytes32 messageHash, bytes calldata signature, bool isUplinkProof) external`
  - `messageHash`: keccak256 hash of the delivered message packet.
  - `signature`: ECDSA signature over `messageHash` by the receiver.
  - `isUplinkProof`: Boolean indicating if the proof is for an uplink message (true) or downlink (false), to determine the expected signer.
  - `external`: Callable only from outside the contract.
  - `calldata`: Efficient data location for external args.

Access Control and Security Checks (`require` statements)

Crucial for security and correctness.

- Prevent Double Claim: `require(!claimedHashes[messageHash], "Reward already claimed").`
- Check Contract Funds: `require(address(this).balance >= fixedRewardAmount, "Insufficient funds").`
- Determine Expected Signer: Based on `isUplinkProof`, set `expectedSigner` to `baseStationAddress` or `iotDeviceAddress`.
- Verify Signature Authenticity: Use `ecrecover` (with Ethereum signed message prefix) to get `signerAddress. require(signerAddress == expectedSigner, "Invalid signature").`

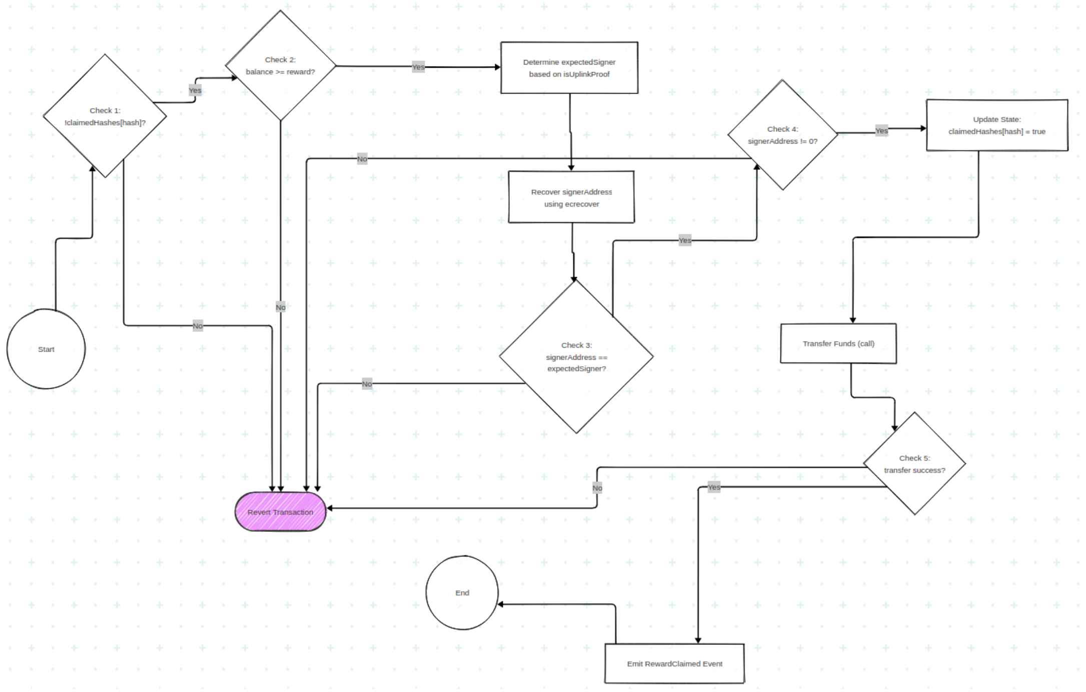

Figure 3.4: Flowchart for claimReward Logic

- Check Signature Validity: `require(signerAddress != address(0), "Signature recovery failed")`.
- Verify Transfer Success: Use low-level `call` for Ether transfer. `(bool success, ) = msg.sender.call{value: fixedRewardAmount}(""); require(success, "Transfer failed");`. (Follows checks-effects-interactions pattern).

Event Emission (`RewardClaimed`)

For off-chain monitoring.

- Definition: `event RewardClaimed(address indexed intermediary, bytes32 indexed messageHash, address indexed signer, uint256 amount);`
- Emission: Emitted on successful `claimReward`.
- Purpose: Logs intermediary, message hash, actual signer, and amount for transparent history.

Funding Mechanism (`receive` function)

To accept Ether for rewards.

- `receive() external payable { }`: Allows direct Ether transfers to the contract address. Simplest method.
- Alternative: Dedicated `deposit()` function if more logic needed.

3.3.2 Off-Chain Proof Generation (Receiver: Base Station/IoT Device)

Generated by the entity that successfully validated the message.

Data Hashing for Proof Integrity

- Input: Entire raw byte sequence of the validated packet.
- Algorithm: `keccak256` (Ethereum standard).
- Output: 32-byte `messageHash`. Ties proof to the specific transmission.

ECDSA Signature Generation using Receiver's Key

- Input: 32-byte `messageHash`.
- Signing Key: Receiver's static private key (sk_B for uplink, sk_I for downlink).
- Algorithm: ECDSA (e.g., NIST P-256/secp256r1).
- Output: ECDSA `signature` (v, r, s components). Proves receiver acknowledged receipt/validation.

3.3.3 Interaction Flow for Reward Claim

Integrates protocol steps with smart contract interaction.

1. Message Transmission Validation (Section 3.2).
2. Receiver Validation (Signature, Decryption/Tag check).
3. Proof Generation (Off-Chain): Receiver computes `messageHash = keccak256(raw_packet)`, `signature = ECDSA_Sign(receiver_sk, messageHash)`.
4. Proof Transfer: Receiver sends (`messageHash`, `signature`) to Intermediary (mechanism assumed).
5. Reward Claim Transaction (On-Chain): Intermediary uses its EOA, calls `claimReward(messageHa` `signature, isUplinkProof)` on contract, pays Gas.
6. Smart Contract Execution (On-Chain): Nodes execute `claimReward` logic (verification checks).
7. Outcome: Success (state updated, reward transferred, event emitted) or Failure (transaction reverts).

Chapter 4

Implementation Details

This chapter provides a detailed account of the practical implementation of the secure, incentivized IoT communication system designed in Chapter 3. It covers the development environment setup, the specific tools and libraries employed, the structure and key logic within the codebase residing in the `Aninda001/IoT-blockchain` repository, and the process for deploying and initializing the smart contract component.

4.1 Development Environment and Technology Stack

Leverages a modern JavaScript and blockchain development stack.

- Node.js Runtime Environment: Core runtime for off-chain components (IoT, Base Station, Intermediary) (e.g., v18.x+). Event-driven I/O suitable for network/async operations.
- Hardhat Framework: Primary development environment for Ethereum smart contract (`MessageReward.sol`). Facilitates compiling, testing (Mocha/Chai), deploying, and local node management (Hardhat Network).
- Ethers.js Library: JavaScript library for blockchain interaction from off-chain components (Intermediary) (e.g., v5.x/6.x). Connects to nodes (JSON-RPC), manages wallets, sends transactions, interacts with contracts.
- Solidity Compiler Version: Specific version used for `MessageReward.sol` (e.g., solc 0.8.19) configured in `hardhat.config.js`. Ensures deterministic bytecode.
- Cryptographic Libraries (`node:crypto`): Built-in Node.js module for off-chain crypto primitives: ECC (key gen, ECDH), ECDSA (sign/verify), AES-GCM (encrypt/decrypt), Hashing (SHA-256, Keccak-256), KDF (HKDF).
- Ganache / Hardhat Network: Local blockchain test environment for development/testing. Hardhat Network often preferred for integration; provides accounts with test Ether.

4.2 Component Implementation (`Aninda001/IoT-blockchain` repository)

Logic implemented across key files.

- `iot-device.js`: Simulates IoT device. Loads keys (sk_I, pk_I, pk_B). Generates ephemeral keys. Performs uplink crypto (ECDH, HKDF, AES-GCM encrypt, SHA256 hash, ECDSA sign). Constructs/sends packet. Handles downlink crypto (ECDSA verify, ECDH, HKDF, AES-GCM decrypt) and proof generation (keccak256 hash, ECDSA sign).
- `base-station.js`: Represents Base Station. Loads keys (sk_B, pk_B, pk_I). Receives uplink packets. Parses/verifies signature (pk_I). Performs ECDH (sk_B, epk_I), HKDF, AES-GCM decrypt. Processes payload. Generates uplink reward proof (keccak256 hash, ECDSA sign with sk_B). Sends proof to Intermediary. Handles downlink message generation symmetrically.
- `intermediary.js`: Simulates untrusted relay. Receives/forwards packets. Receives reward proof (hash, signature). Loads own EOA private key. Uses Ethers.js to connect to network, instantiate contract object (address, ABI), call `claimReward(hash, signature, isUplinkProof)`, handles transaction submission/logging.
- `crypto_utils.js`: Encapsulates common crypto operations (wrappers around `node:crypto`) for modularity. Functions like `generateEccKeyPair()`, `computeEcdhSecret()`, `deriveKeysHkdf()`, `encryptAesGcm()`, `decryptAesGcm()`, `signEcdsa()`, `verifyEcdsa()`, `hashSha256()`, `hashKeccak256()`.
- `contracts/MessageReward.sol`: Solidity source code. Includes version pragma, imports (e.g., OpenZeppelin ECDSA), state variables, events, constructor, `claimReward` function (validation, ecrecover, state updates, transfer, event), `receive()` function. (Refer to Appendix [X] for full code).
- `config.js, .env`: Configuration management. `.env` stores sensitive data (private keys, RPC URLs, reward amount) - not version controlled. `config.js` loads from `.env` (using `dotenv`), provides structured access to parameters for other scripts.

4.3 Smart Contract Deployment and Initialization

Using Hardhat Ignition for robust, declarative deployments.

- Hardhat Ignition Modules (`ignition/modules/deploy.js`): Defines deployment plan declaratively. Specifies contract (`MessageReward`), constructor arguments (from config), dependencies.
- Deployment Script (`ignition/deploy.js` or Hardhat task): Orchestrates deployment. Uses Hardhat Runtime Environment (hre), Ethers.js, Ignition. Specifies target network. Invokes Ignition's `deploy` function with the module. Logs deployed contract address.
- Automated Contract Funding Logic: Handled by deployment script post-deployment. Retrieves deployed address. Determines funding amount (from config). Uses Ethers.js

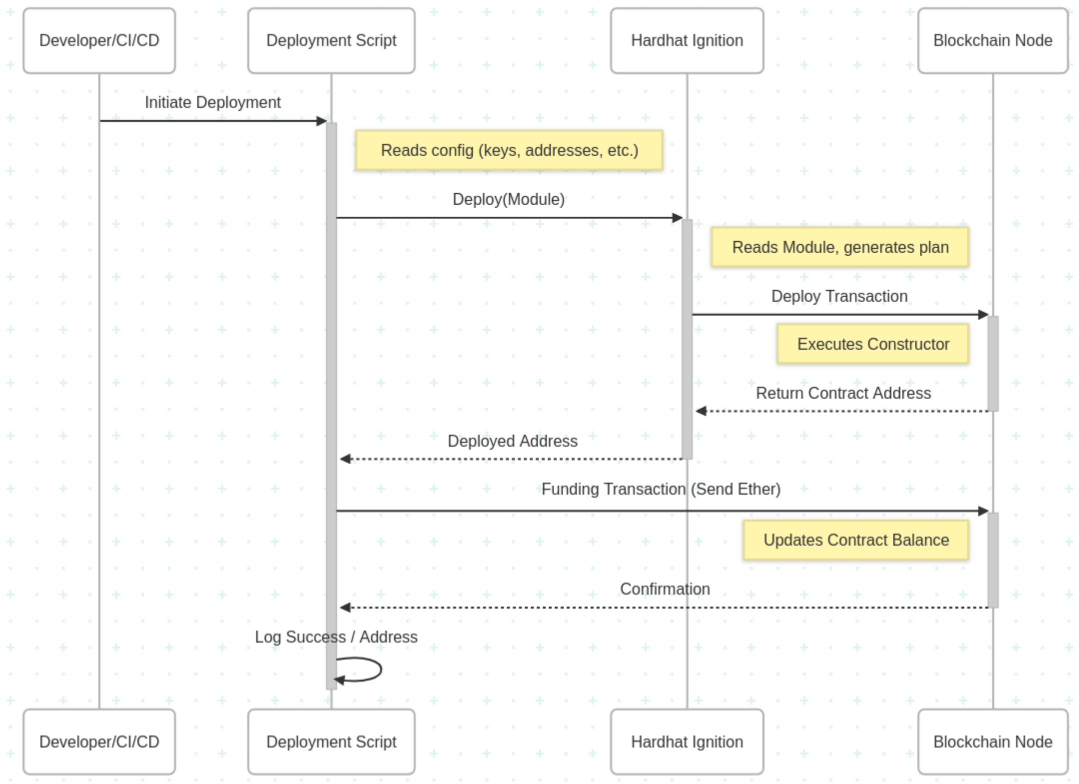

Figure 4.1: Deployment & Funding Sequence Diagram

signer (deployer account) to send Ether transaction to contract address (using `receive()`). Waits for confirmation, logs result. Ensures contract is operational immediately.

Chapter 5

Design Analysis and Discussion

5.1 Introduction

This chapter presents a critical analysis of the system designed in Chapter 3 and implemented as described in Chapter 4. This analysis is primarily theoretical, based on architectural design, cryptographic principles, and smart contract logic, as comprehensive empirical testing was not performed.

Disclaimer: This analysis does not present measured experimental results. Empirical validation (performance, energy, network tests) is required for definitive conclusions about practical feasibility.

The analysis focuses on:

- Security Analysis (Section 5.2): Resilience against the threat model.
- Theoretical Performance Analysis (Section 5.3): Expected computational, communication, and blockchain costs.

Section 5.4 provides a broader Discussion, interpreting findings, acknowledging limitations, and suggesting future work.

5.2 Security Analysis

Evaluates security properties against the threat model (Section 3.1.2), assuming standard crypto algorithm security and secure static key pre-distribution.

5.2.1 Confidentiality Analysis (Against Eavesdropping)

- Threat: Attacker/Intermediary learns payload content.
- Defense: End-to-end AES-128-GCM encryption. Session key derived via ephemeral ECDH, known only to endpoints.
- Analysis: Secure. Eavesdropper cannot derive AES key without private keys (ephemeral or static). Intermediary sees only opaque ciphertext. Provides E2E confidentiality.

5.2.2 Integrity Analysis (Against Tampering - AES-GCM Tag)

- Threat: Attacker modifies packet content (ciphertext, epk) without detection.
- Defense: AES-GCM AEAD generates authentication tag (auth_tag) covering ciphertext and AAD (epk). Receiver recalculates and verifies tag during decryption.
- Analysis: Secure. Modification of ciphertext or epk leads to tag mismatch at receiver with overwhelming probability. Decryption fails, packet rejected. Ensures data integrity and AAD authenticity. Tampering is detectable.

5.2.3 Authenticity Analysis (ECDSA Signatures)

- Threat: Attacker impersonates legitimate endpoint by injecting forged packets.
- Defense: Each packet signed with sender's static private key (sk_I or sk_B) using ECDSA. Signature covers hash(epk || ciphertext || tag). Receiver verifies using sender's known static public key (pk_I or pk_B).
- Analysis: Secure. Attacker lacking static private key cannot forge valid signature. Receiver verification ensures packet originated from claimed sender. Provides source authenticity.

5.2.4 Replay Attack Resistance (Ephemeral Keys/Nonces & Contract State)

- Threat: Attacker captures valid packet and retransmits later.
- Defense:
  - Protocol Level: Ephemeral keys ensure unique session keys. AES-GCM nonce usage (derived via HKDF) prevents reuse within a session context. (Higher-level sequence numbers could add further protection if needed).
  - Reward Mechanism Level: Smart contract's `claimedHashes` mapping tracks hashes of rewarded messages. `require(!claimedHashes[messageHash])` prevents claiming reward twice for the same hash.
- Analysis: Robust against reward replay due to smart contract state. Protocol provides PFS but relies on nonce uniqueness or higher-level state for transport replay detection. Smart contract is primary defense against reward fraud via replay.

5.2.5 Man-in-the-Middle (MitM) Attack Resilience

- Threat: Intermediary/attacker intercepts and relays messages, potentially impersonating endpoints.
- Defense: Mutual authentication via ECDSA signatures based on pre-shared static public keys. ECDH exchange bound to authenticated identity. E2E encryption.
- Analysis: Secure. MitM cannot forge ECDSA signatures without static private keys. Cannot read payload due to E2E encryption. Protocol functions securely despite intermediary being a potential MitM.

5.2.6 Smart Contract Security Considerations

- Threat: Vulnerabilities in `MessageReward.sol` leading to fund theft, incorrect rewards, DoS.
- Defense Analysis:
  - Signature Verification: Relies on `ecrecover`. Correct usage (Ethereum prefix) and comparison against stored receiver address (`baseStationAddress` or `iotDeviceAddress`) is crucial and standard practice. Secure if implemented correctly.
  - Double-Spending Prevention: `claimedHashes` mapping and check effectively prevent reward replay.
  - State Consistency: Checks-effects-interactions pattern followed (mark claimed before transfer). Use of low-level `call` mitigates reentrancy.
  - Access Control: Constructor initializes immutable state. Admin functions would need proper controls (e.g., `Ownable`).
  - Gas Limits DoS: `claimReward` cost dominated by `ecrecover` and `SSTORE`. Predictable cost. Contract running out of funds causes economic DoS for legitimate claims (importance of funding).
  - Trust: Relies on correctness of deployed code and security of underlying blockchain.

Summary: Security analysis indicates strong theoretical guarantees for confidentiality, integrity, authenticity, MitM resilience, and reward replay protection, assuming correct implementation and secure key management.

5.3 Theoretical Performance Analysis

Analyzes expected performance characteristics (computational, communication, blockchain costs) without empirical measurements.

5.3.1 Analysis of Cryptographic Operation Costs (Relative Intensity)

- Asymmetric (ECC - P-256):
  - ECDH Key Gen (Ephemeral): Moderate cost (scalar mult). Sender only.
  - ECDH Secret Comp: Moderate/High cost (scalar mult). Sender Receiver.
  - ECDSA Sign: Intensive (hash, RNG, scalar mult). Sender (msg), Receiver (proof). Most expensive.
  - ECDSA Verify: Less intensive than sign, still significant (multi-scalar mult). Receiver (msg), Contract (`ecrecover`).
- Symmetric (AES-128-GCM): Highly optimized. Cost proportional to payload size. Much faster than ECC.
- Hashing/KDF (SHA-256, Keccak-256, HKDF): Very fast, negligible compared to ECC/AES.

Relative Cost Summary: ECDSA Sign > ECDSA Verify $\approx$ ECDH Ops > AES-GCM (per byte) > Hashing/HKDF. **Implications:** ECC operations are the main computational burden for endpoints. Device efficiency depends heavily on hardware crypto support.

5.3.2 Communication Overhead Calculation

Security protocol adds fixed overhead to application payload. Assumptions: NIST P-256, Uncompressed points (65 bytes), AES-128-GCM tag (16 bytes), ECDSA signature ($\sim$71 bytes). Overhead = Size(epk) + Size(auth_tag) + Size(signature) = 65 + 16 + 71 = **152 bytes**. (Excludes transport headers).

Table 5.1: Theoretical Communication Overhead Analysis

Payload Size (P)	Security Overhead (O)	Total Size (P + O)*	Overhead Percentage
16 bytes	152 bytes	168 bytes	90.5%
32 bytes	152 bytes	184 bytes	82.6%
64 bytes	152 bytes	216 bytes	70.4%
128 bytes	152 bytes	280 bytes	54.3%
256 bytes	152 bytes	408 bytes	37.3%
512 bytes	152 bytes	664 bytes	22.9%

* Note: Excludes underlying transport protocol headers.

Analysis: Fixed overhead (152 bytes) is substantial, especially for small payloads (>90% for 16 bytes). Relative overhead decreases with larger payloads. Critical trade-off: security vs. packet size. Challenging for bandwidth-constrained networks (LPWANs). Compressed keys ($\sim$33 bytes) offer potential optimization ($\sim$32 bytes saving).

5.3.3 Estimated Blockchain Transaction Costs

Gas costs incurred for contract interactions.

- Contract Deployment: One-time, high cost (bytecode size + constructor SSTOREs).
- `claimReward` Call: Recurring cost (paid by Intermediary). Factors:
  - Base Tx Cost: Medium.
  - SLOAD (read state): Medium.
  - `ecrecover` (precompile): High.
  - SSTORE (write `claimedHashes`): Very High (dominant cost).
  - CALL (Ether transfer): Medium.
  - LOG (event): Medium.
  - Other ops: Low.

Analysis: `claimReward` incurs non-trivial gas costs, dominated by SSTORE and ecrecover. Economic viability depends on Reward Amount vs. Gas Cost (influenced by Ether price, network congestion). Reward must significantly outweigh cost for incentive to work. Layer 2 solutions could drastically reduce costs.

Table 5.2: Estimated Relative Gas Cost Factors for claimReward Operations

Operation within claimReward	Typical Relative Gas Cost
Base Transaction Fee	Medium
SLOAD (read claimedHashes, etc.)	Medium
ecrecover (Precompile)	High
SSTORE (write claimedHashes=true)	Very High
CALL (Ether transfer)	Medium
LOG (Emit RewardClaimed event)	Medium
Other operations (memory, compares)	Low

5.3.4 Performance Implications for Resource-Constrained Devices

Synthesizing the impact on IoT devices.

- **Latency:** ECC operations introduce latency (depends on HW acceleration). May be unacceptable for time-critical apps.
- **Energy Consumption:** ECC computations and transmission of larger packets consume significant energy. Critical for battery life.
- **Memory Usage:** Crypto libraries, keys, intermediate values require RAM/Flash.
- **Communication Constraints:** High overhead ($\sim$152 bytes) may exceed LPWAN packet limits or consume excessive bandwidth/energy.
- **Blockchain Interaction Burden:** Offloaded from IoT device, but device still performs hash/sign for downlink proof generation.

Conclusion on Implications: Significant security-performance trade-off. Robust E2E security imposes considerable computational (ECC) and communication overhead. May be challenging for severely constrained devices/networks. Potentially acceptable for more capable devices (e.g., ESP32+) and less restrictive networks where security/incentives are prioritized. Suitability requires careful consideration of specific application/device/network constraints.

5.4 Discussion

Synthesizes findings, interprets implications, acknowledges limitations, proposes future work.

5.4.1 Interpretation of Analytical Findings

- **Assessed Effectiveness of Hybrid Encryption Protocol:** Theoretically effective. Achieves E2E confidentiality (AES-GCM), integrity (AES-GCM tag), authenticity (ECDSA), and PFS (ephemeral ECDH). Robust against threats within the model.
- **Assessed Viability of Blockchain-Based Reward Mechanism:** Functionally viable. Secure proof validation (ecrecover), double-spending prevention (`claimedHashes`). Economic viability depends critically on Reward vs. Gas Cost ratio.
- **Security vs. Performance Trade-offs Discussion:** Fundamental trade-off. Robust security requires intensive ECC ops and high communication overhead. Blockchain adds

transaction costs. Acceptability depends on application context (data value vs. latency/energy/bandwidth needs).

5.4.2 Limitations of the Study

- **Lack of Empirical Validation:** Most significant limitation. Theoretical analysis needs confirmation with real hardware/network measurements (latency, energy, throughput, actual gas costs).
- **Network Assumptions and Simplified Models:** Assumes relatively ideal network. Doesn't deeply model packet loss, latency variations. Reward proof transfer mechanism simplified. Trust model simplifies some aspects (e.g., receiver always generates proof).
- **Scope of Security and Performance Analysis:** Focused on network threats and core contract logic. Excludes physical security, side-channels, advanced contract/game-theoretic attacks. Performance analysis theoretical (no energy profiling, memory footprint).

5.4.3 Future Work and Enhancements

Addressing limitations and building upon the design.

- **Implementation on Physical Hardware and Empirical Testing:** Highest priority. Measure latency, energy, throughput, memory, gas costs on representative platforms (ESP32, etc.) and networks.
- **Integration with Standard IoT Protocols:** Integrate with MQTT or CoAP for easier deployment and evaluation in standard frameworks.
- **Exploration of Alternative Blockchain Solutions:** Evaluate Layer 2s (Optimism, Arbitrum) or other L1s (Solana, Polygon) to reduce gas costs. Consider permissioned ledgers if applicable.
- **Advanced Smart Contract Features:** Dynamic rewards, reputation/staking/slashing, scalability patterns (registries, merkle proofs), upgradeability (proxies).
- **Formal Security Verification Methods:** Apply formal methods (model checking, theorem proving) to rigorously verify contract logic and protocol properties.
- **Scalability Testing:** Simulate or test with larger numbers of devices/intermediaries to evaluate behavior under load and identify bottlenecks.
- **Cryptographic Optimizations:** Investigate alternative curves (Curve25519), schemes (IBE), or compressed public keys to reduce overhead/computation.

Chapter 6

Conclusion

This thesis addressed the challenge of secure, incentivized E2E communication for IoT devices over untrusted intermediaries.

6.1 Summary of Research and Contributions

Proposed, designed, implemented (simulated), and analyzed a system integrating hybrid cryptography with a blockchain reward layer.

- **Summary of Proposed System:**
  - Hybrid E2E Encryption Protocol: ECDH (ephemeral) + AES-GCM + ECDSA (static) for confidentiality, integrity, authenticity, PFS over untrusted intermediaries.
  - Blockchain Reward Mechanism: `MessageReward.sol` smart contract for trustless, automated reward disbursement based on receiver-signed cryptographic proof (hash + signature).
  - Defined System Roles: IoT Device, Base Station, Intermediary.
- **Key Contributions:**
  - Novel System Design: Integrated architecture combining tailored E2E crypto with blockchain micro-incentives.
  - Component Implementation: Functional prototype (Node.js, Solidity/Hardhat).
  - Comprehensive Security Analysis: Theoretical evaluation against relevant threats.
  - Theoretical Performance Evaluation: Assessment of computational, communication, blockchain overheads.
  - Identification of Trade-offs and Future Directions: Articulation of security-performance balance, limitations, and future work.

6.2 Concluding Remarks on the Proposed System

Theoretical analysis indicates the system successfully addresses core requirements: robust E2E security via the hybrid protocol and a functionally sound, verifiable reward mechanism via the smart contract.

However, significant performance trade-offs exist: security imposes computational (ECC) and communication overhead, and blockchain adds transaction costs. Practical economic feasibility depends on the reward vs. gas cost balance.

Potential Applicability: Well-suited for IoT apps where E2E security is paramount, communication involves untrusted intermediaries, explicit incentives are needed, and devices/networks can tolerate the overhead. Less suitable for extreme low-latency/power/bandwidth scenarios or where data value doesn't justify transaction costs.

Final Perspective: This research presents a specific design integrating modern crypto with blockchain for secure, incentivized IoT communication. It demonstrates potential while acknowledging performance implications. Lack of empirical validation necessitates future work, but this study provides a solid foundation and blueprint. Insights contribute to exploring secure, self-sustaining IoT communication paradigms. Bridging the gap to practical deployment via empirical evaluation is the essential next step.

Bibliography

[1] Robin Chataut, Alex Phoummalayvane, Robert Akl, "A Comprehensive Review of IoT Applications and Future Prospects," *Sensors*, Vol. 23, No. 16, pp. 7194, 2023.

[2] Market Research Firm, "Global IoT Market Forecasts 2025-2030," 2025. Available at: https://www.statista.com/outlook/tmo/internet-of-things/worldwide

[3] Asimily, "The Top Internet of Things (IoT) Cybersecurity Breaches in 2024," 2024. Available at: https://asimily.com/blog/the-top-internet-of-things-iot-cybersecurity-breaches-in-2024/

[4] A. Chehab et al., "Performance and overhead evaluation of OSCOAP and DTLS," *CORE*, 2020. Available at: https://core.ac.uk/download/301010414.pdf

[5] Twingate, "What is Network Sniffing? How It Works Examples," 2023. Available at: https://www.twingate.com/blog/glossary/network%20sniffing

[6] K. M. Jain et al., "A Survey on Man in the Middle Attack," *IJSTE*, Vol. 2, No. 9, 2016. Available at: http://www.ijste.org/articles/IJSTEV2I9103.pdf

[7] Packet Labs, "A Guide to Replay Attacks And How to Defend Against Them," 2023. Available at: https://www.packetlabs.net/posts/a-guide-to-replay-attacks-and-how-to-defend-against-them/

[8] Tripwire, "How to Leverage NIST Cybersecurity Framework for Data Integrity," 2023. Available at: https://www.tripwire.com/state-of-security/how-leverage-nist-cybersecurity-framework-data-integrity

[9] IETF RFC 9147, "The Datagram Transport Layer Security (DTLS) Protocol Version 1.3," 2022. Available at: https://datatracker.ietf.org/doc/html/rfc9147

[10] A. Nofal, "Analyzing and Reducing the Overhead of TLS and DTLS in WiFi-Based IoT Networks," PhD Thesis, Santa Clara University, 2022. Available at: https://scholarcommons.scu.edu/eng_phd_theses/41

[11] H. V. Chand et al., "Analyzing Power Consumption of TLS Ciphers on an ESP32," 2022. Available at: https://www.wolfssl.com/how-much-battery-power-does-tls-use/

[12] A. Nofal, "eeDTLS: Energy-Efficient Datagram Transport Layer Security for the Internet of Things," 2022. Available at: https://arxiv.org/pdf/2011.12035

[13] IETF RFC 9147, "The Datagram Transport Layer Security (DTLS) Protocol Version 1.3," 2022. Available at: https://www.ietf.org/archive/id/draft-ietf-tls-dtls13-35.html

[14] N. Koblitz, "Elliptic curve cryptosystems," *Mathematics of Computation*, 1987. / V. Miller, "Use of elliptic curves in cryptography," *CRYPTO '85*, 1986.

[15] D. J. Bernstein, "Security Analysis of Elliptic Curves over Sextic Extension of Small Fields," *IACR Cryptology ePrint Archive*, 2022. Available at: https://eprint.iacr.org/2022/277.pdf.

[16] NIST FIPS PUB 186-4, "Digital Signature Standard (DSS)," 2013. / SECG, "SEC 2: Recommended Elliptic Curve Domain Parameters," 2010.

[17] D. J. Bernstein, "Security dangers of the NIST curves," *2013*, Available at: https://cr.yp.to/talks/2013.05.31/slides-dan+tanja-20130531-4x3.pdf.

[18] NIST FIPS PUB 197, "Advanced Encryption Standard (AES)," 2001.

[19] NIST Special Publication 800-38D, "Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC," 2007.

[20] IETF RFC 5869, "HMAC-based Extract-and-Expand Key Derivation Function (HKDF)," 2010.

[21] S. Nakamoto, "Bitcoin: A Peer-to-Peer Electronic Cash System," 2008.

[22] V. Buterin, "Ethereum White Paper: A Next-Generation Smart Contract and Decentralized Application Platform," 2014.

[23] G. Wood, "Ethereum: A Secure Decentralised Generalised Transaction Ledger (Yellow Paper)," 2014 (and updates).

[24] Coinbase, "What is 'proof of work' or 'proof of stake'?" *Coinbase*, 2023. Available at: https://www.coinbase.com/learn/crypto-basics/what-is-proof-of-work-or-proof-of-stake.

[25] MoonPay, "What are Ethereum gas fees? ETH fees explained," 2023. Available at: https://www.moonpay.com/learn/defi/what-are-ethereum-gas-fees.

[26] N. Szabo, "Smart Contracts: Building Blocks for Digital Markets," 1996.

[27] Solidity Language Documentation, https://docs.soliditylang.org/.

[28] IETF RFC 4279, "Pre-Shared Key Ciphersuites for Transport Layer Security (TLS)," 2006. / IETF RFC 6347, "Datagram Transport Layer Security Version 1.2," 2012 (includes PSK).

[29] IETF RFC 7250, "Using Raw Public Keys in Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)," 2014.

[30] Relevant sections of TLS RFCs (e.g., RFC 5246, RFC 8446) describing session resumption.

[31] IETF RFC 9147, "The Datagram Transport Layer Security (DTLS) Protocol Version 1.3," 2022 (covers CID).

[32] IETF RFC 8613, "Object Security for Constrained RESTful Environments (OSCORE)," 2019.

[33] LoRa Alliance, "LoRaWAN Specification v1.0.4," 2020. Available at: https://www.thethingsnetwork.org/docs/lorawan/what-is-lorawan/.

[34] H. Liu et al., "Blockchain-enabled access control to prevent cyber attacks in IoT," *Frontiers in Big Data*, 2022. Available at: https://www.frontiersin.org/journals/big-data/articles/10.3389/fdata.2022.1081770/full.

[35] L. Ma et al., "A systematic review on blockchain-based access control systems in IoT," *Journal of Cloud Computing*, 2024. Available at: https://journalofcloudcomputing.springeropen.com/articles/10.1186/s13677-024-00697-7.

[36] A. A. Almazroi et al., "Blockchains for industrial Internet of Things in sustainable supply chain management," *Sustainable Supply Chain Management*, 2024. Available at: https://www.sciencedirect.com/science/article/pii/S2667344424000094.

[37] S. Pal et al., "Blockchain-Enabled Secure Decentralized Identity Management for IoT," *SSRN*, 2023. Available at: https://papers.ssrn.com/sol3/papers.cfm?abstract_id=4904931.

[38] Filecoin Project, "A Decentralized Storage Network," 2021. Available at: https://filecoin.io/filecoin.pdf.

[39] Storj Labs, "Storj Documentation," 2023. Available at: https://storj.dev/.

[40] Helium Network Documentation, https://docs.helium.com/.

[41] A. Popa et al., "On Maximizing Collaboration in Wireless Mesh Networks Without Monetary Incentives," *WiOpt*, 2010. Available at: https://inria.hal.science/inria-00502014v1/document.