Blockchain-Incentivized Secure Communication for Trustless IoT Relay Networks

---

Aninda Nath, Gaurav Kumar Sharma, Dwaipayan Biwas, Shibam Nath

8 May 2025

Supervisor: Dr. Niladri Das
Indian Institute of Engineering Science and Technology, Shibpur

Agenda

Motivation & Challenges

Problem Definition & Approach

System Architecture

System Design (Security Protocol, Blockchain Incentive)

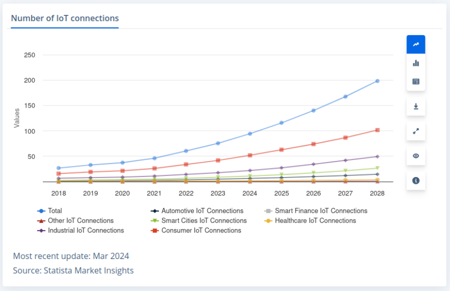
Discussion & Future Work

Conclusion

Motivation & Challenges

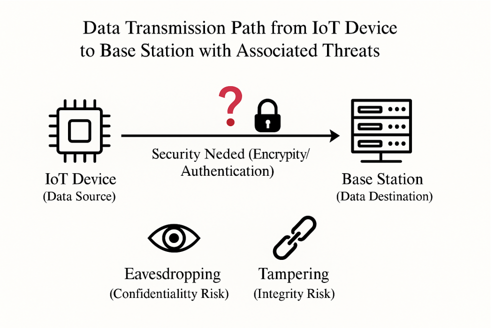
The Growth of IoT

- **Rapid Market Expansion:** Strong indicator of increasing device deployment. Projected worldwide revenue of **US$1.06tn by 2025**.
- **Significant Growth Rate:** Anticipated annual growth of **10.17% (CAGR 2025-2029)**, leading to a projected market volume of **US$1.56tn by 2029**.



- **Diverse Sector Adoption:** Industrial IoT projected as the largest segment (**US$275.70bn in 2025**); continued investment in areas like **Smart Homes** and **Connected Cars**.
- **Massive Data Generation:** A direct consequence of IoT expansion, driving further innovation and infrastructure needs.

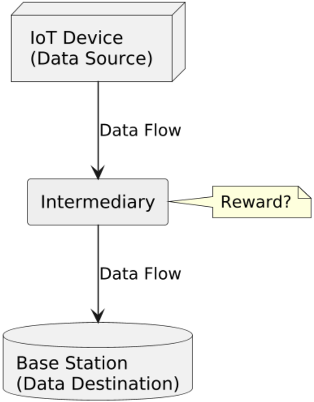
IoT Challenges: Security & Trust



- **Need: End-to-End Security:** Essential to protect data from source to destination.
  - *Confidentiality:* Prevent eavesdropping.
  - *Integrity:* Ensure data isn't altered.
  - *Authenticity:* Verify data origin.
- **Massive Scale & Data Sensitivity:** Billions of devices generating potentially critical or private information.
- **Challenge: Untrusted Networks:** Data often travels across public internet or third-party infrastructure, increasing exposure to attacks.

Challenge: Untrusted Relays & Incentives

- **Reliance on 3rd Party Relays:** IoT communication often depends on intermediate nodes (gateways, brokers) to forward data.
- **Trust & Motivation Issues:** These relays may be untrusted, unreliable, or lack economic incentive to consistently forward data. They could be offline, overloaded, or even malicious.
- **Core Question:** How can we ensure reliable data forwarding when we don't control or fully trust the intermediary?
- **Need: Verifiable Incentive Mechanism:** A system is required to motivate relays and provide verifiable proof of their service, ensuring they get rewarded *only* for successful forwarding.



Problem Definition & Approach

Problem Definition & Approach

Problem Definition

How to achieve End-to-End Secure (Confidential, Integrity-Protected, Authenticated) communication between IoT Devices and a Base Station via Untrusted Intermediaries, while providing a Trustless, Verifiable, and Automated Incentive Mechanism for reliable relay?

Our Approach

- **A Hybrid Encryption Protocol** designed and implemented using robust cryptographic techniques suitable for constrained devices.
- **A Blockchain-Based Smart Contract** implemented to provide verifiable incentives through a transparent and automated reward system.

System Architecture

Related Work & Contribution

Gap in Existing Work:

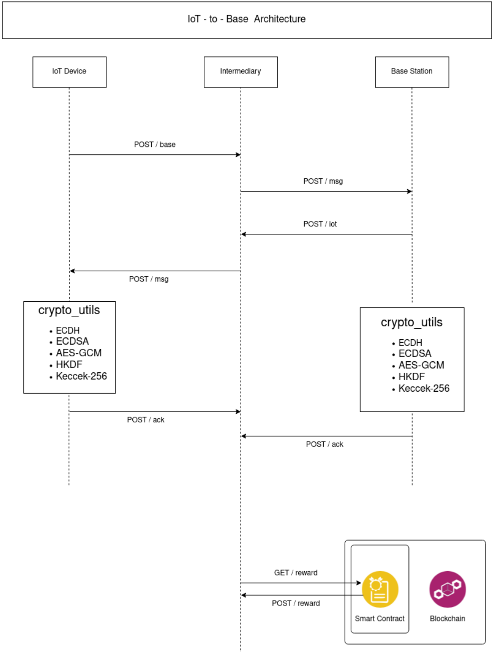
- While solutions exist for IoT security or incentive mechanisms separately, there is a lack of integrated solutions addressing both End-to-End (E2E) security and trustless relay incentives simultaneously, especially for communication via untrusted intermediaries.

Our Contribution:

- We present a **novel framework** that uniquely combines tailored cryptographic protocols for E2E security with **blockchain-based rewards** for verifiable relay incentives.
- This provides a **holistic solution** specifically designed for secure and reliable data transmission in resource-constrained IoT environments with untrusted relays.

Proposed System Architecture

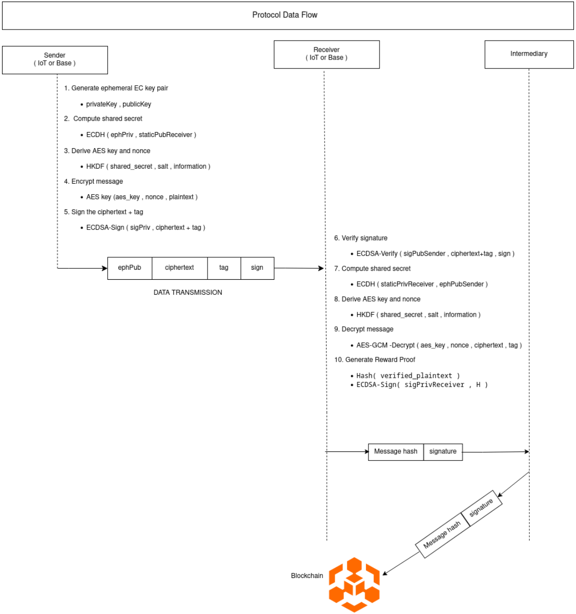
- **Key Actors:**
  - IoT Device (Data Source)
  - Untrusted Intermediary (Relay)
  - Base Station (Data Destination)
- **End-to-End Security:** Data is encrypted between the Device and Base Station; the Intermediary handles only opaque (encrypted) data.
- **Incentive Layer:** The Blockchain acts as the Trust Anchor, executing the Smart Contract for the Reward Logic.



System Roles & High-Level Flow

Roles & Responsibilities

- **IoT Device / Base Station:**
  - Encrypt & Sign outbound data.
  - Decrypt & Verify inbound data.
  - Generate Reward Proofs upon successful data receipt.
- **Intermediary (Relay):**
  - Relay encrypted packets (cannot read content).
  - Submit Reward Proofs to the Blockchain Smart Contract.
  - Receive rewards upon successful verification by the contract.



System Design

Security Protocol: ECDH + HKDF + AES-GCM + ECDSA

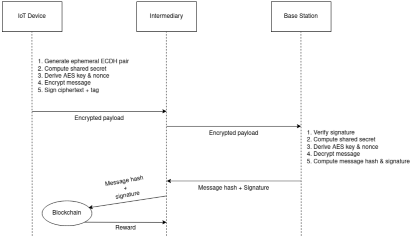
Hybrid Cryptographic Approach:

- **Elliptic Curve Diffie-Hellman (ECDH):**
  - Used for secure Session Key Exchange.
  - Employs Ephemeral Keys (per-session) $\rightarrow$ Forward Secrecy.
  - Generates a shared secret.
- **HMAC-based Key Derivation Function (HKDF):**
  - Derives strong cryptographic keys (e.g., AES key, nonce) from the ECDH shared secret, salt, and context information.
- **AES with Galois/Counter Mode (AES-GCM):**
  - Provides Authenticated Encryption using the HKDF-derived session key.
  - Ensures Confidentiality & Integrity.
- **Elliptic Curve Digital Signature Algorithm (ECDSA):**
  - Used for Digital Signatures with Static Keys.
  - Guarantees Sender Authenticity & Non-repudiation.

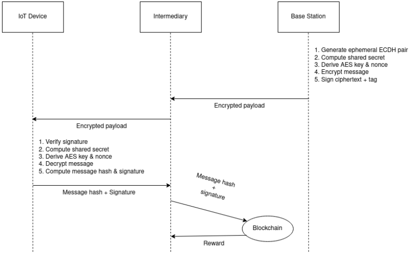
Key Management:

- Static Keys (Long-term): Used for ECDSA (identity).
- Ephemeral Keys (Per-session): Used for ECDH (forward secrecy).
- Session Keys (Derived): Symmetric AES keys derived via ECDH + HKDF, used for AES-GCM.

Communication Flow (Device $\leftrightarrow$ Base)



Uplink Flow



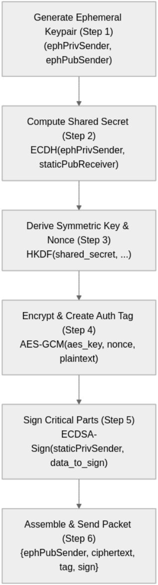
Downlink Flow

- Uplink: IoT Device $\rightarrow$ Base (Base generates proof)
- Downlink: Base $\rightarrow$ IoT Device (IoT Device generates proof)
- Intermediary relays packet unmodified.

Core Crypto Actions: Sender (Device/Base)

Preparing an Encrypted & Signed Packet:

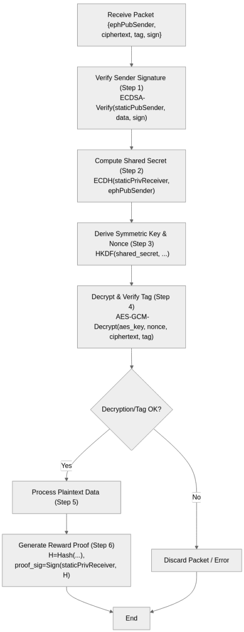
- **Generate Ephemeral Keypair:** Create a fresh, one-time (`ephPrivSender`, `ephPubSender`) keypair for this session (ensures Forward Secrecy).
- **Compute Shared Secret:** Perform ECDH using own ephemeral private key and receiver's static public key: `shared_secret = ECDH(ephPrivSender, staticPubReceiver)`.
- **Derive Symmetric Key:** Use HKDF to derive the AES key and nonce from the `shared_secret`, salt, and context info: `(aes_key, nonce) = HKDF(...)`.
- **Encrypt & Authenticate:** Encrypt the plaintext message using AES-GCM with the derived key and nonce, generating ciphertext and authentication tag.
- **Sign Packet Components:** Create a digital signature using the sender's static private key over critical packet elements (e.g., `ephPubSender + ciphertext + tag`) via ECDSA: `sign = ECDSA-Sign(staticPrivSender, ...)`.
- **Assemble & Send:** Combine `ephPubSender`, `ciphertext`, `tag`, and `sign` into the final packet for transmission.



Core Crypto Actions: Receiver (Base/Device)

Processing an Incoming Packet:

- **Verify Sender Signature:** Use the sender's static public key (`staticPubSender`) to verify the ECDSA sign on the received packet components (`ephPubSender + ciphertext + tag`). → Confirms sender identity and integrity of the signed parts.
- **Compute Shared Secret:** Perform ECDH using own static private key and sender's ephemeral public key: `shared_secret = ECDH(staticPrivReceiver, ephPubSender)`.
- **Derive Symmetric Key:** Use HKDF to derive the AES key and nonce from the `shared_secret`, salt, and context info (must match sender's derivation): `(aes_key, nonce) = HKDF(...)`.
- **Decrypt & Verify Integrity:** Attempt to decrypt the `ciphertext` using AES-GCM with the derived key, nonce, and the received authentication `tag`. → Ensures confidentiality and message integrity.
- **Process Data (If Valid):** If decryption and tag verification succeed, process the resulting `plaintext` data.
- **Generate Reward Proof (If Valid):** Create proof for the intermediary by hashing the relevant data (e.g., `H = Hash(verified_plaintext)`) and signing the hash with the receiver's static private key: `proof_sig = ECDSA-Sign(staticPrivReceiver, H)`. The pair (`H`, `proof_sig`) is made available.



Incentive Mechanism: Blockchain Smart Contract

Automated & Verifiable Rewards:

- The system utilizes a blockchain-based smart contract to manage rewards for Intermediaries, ensuring fairness and reliability.

Leveraging Blockchain Advantages:

- **Immutability:** Reward transactions, once confirmed, cannot be altered.
- **Transparency:** Reward logic and transaction history are publicly auditable on the blockchain.
- **Automation:** Rewards are distributed automatically by the contract upon verification, eliminating manual intervention.

Core Component: `MessageReward.sol`

- This dedicated smart contract encapsulates the reward logic, holds the incentive funds, and processes reward claims.

Trust-Minimized Operation:

- Reliance is placed on the cryptographic proofs generated by devices/base stations and the deterministic execution of the smart contract code, minimizing the need to trust individual actors.

Smart Contract Structure

- **State:**
  - Stores trusted `baseStationAddress` & `iotDeviceAddress`.
  - Defines `fixedRewardAmount`.
  - `claimedHashes` mapping: Prevents replay attacks / double spending.

```
contract MessageReward{

    using ECDSA for bytes32;

    address public baseStationAddress;
    address public iotDeviceAddress;
    uint256 public fixedRewardAmount; // Reward amount in Wei

    // Tracks claimed message hashes to prevent replay attacks
    mapping(bytes32 => bool) public claimedHashes;

    event RewardClaimed(bytes32 indexed messageHash, address indexed intermediary, address indexed signer);
    event ConfigUpdated(address baseStation, address iotDevice, uint256 rewardAmount);

    // --- Constructor & Configuration ---
    constructor(address _initialBaseStation, address _initialIotDevice, uint256 _initialRewardWei) {
        require(_initialBaseStation != address(0), "Invalid Base Station address");
        require(_initialIotDevice != address(0), "Invalid IoT Device address");
        baseStationAddress = _initialBaseStation;
        iotDeviceAddress = _initialIotDevice;
        fixedRewardAmount = _initialRewardWei;
        emit ConfigUpdated(_initialBaseStation, _initialIotDevice, _initialRewardWei);
    }

    function claimReward(bytes32 _messageHash, bytes memory _signature) external {
        // ClaimReward logic
    }
}
```

- **Function** (`claimReward`)**:**
  - Intermediary calls with (`hash`, `isUplink`, `signature`).
  - Verifies signature against correct address (Base/Device).
  - Checks `claimedHashes`.
  - Transfers reward if valid & unclaimed.
- **Events:**
  - `RewardClaimed` : Logs successful claims (`hash`, `intermediary`, `signer`).
  - `ConfigUpdated` : Logs initial setup/updates.

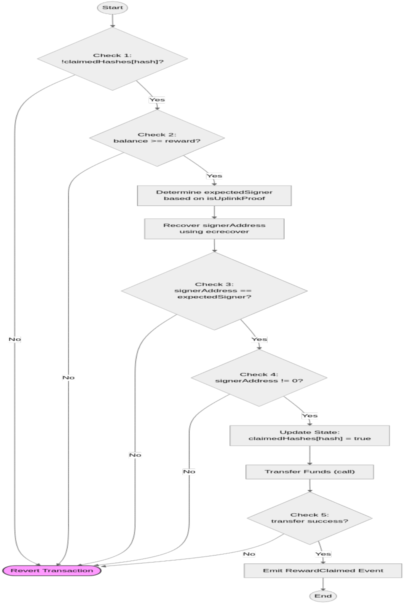
Reward Proof & Claim Flow

Claiming Rewards:

i. **Receiver Generates Proof:**
   - Validates packet, calculates `H = Hash(data)`.
   - Signs hash: `proof_sig = Sign(staticPrivReceiver, H)`.
   - Provides (`H`, `proof_sig`) to Intermediary.

ii. **Intermediary Submits Claim:**
   - Calls `contract.claimReward(H, isUplink, proof_sig)`.

iii. **Contract Verifies & Pays:**
   - Checks `proof_sig` is valid from expected signer (Base/Device).
   - Checks `H` is unique (not in `claimedHashes`).
   - If valid: Transfers reward to Intermediary, marks `H` as claimed.



claimReward Logic

```
function claimReward(bytes32 _messageHash, bytes memory _signature) external {
        require(fixedRewardAmount > 0, "Reward amount not set");
        require(!claimedHashes[_messageHash], "Reward already claimed for this hash");

        // Verify the signature against the hash
        bytes32 prefixedHash = keccak256(abi.encodePacked("\x19Ethereum Signed Message:\n32", _messageHash));
        address recoveredSigner = ECDSA.recover(prefixedHash, _signature);

        require(recoveredSigner != address(0), "Invalid signature");

        // Check if the signer is one of the authorized parties
        require(recoveredSigner == baseStationAddress || recoveredSigner == iotDeviceAddress, "Signer not authorized");

        // Mark as claimed BEFORE transfer to prevent re-entrancy style issues
        claimedHashes[_messageHash] = true;

        // Transfer fixed reward to the intermediary (msg.sender)
        (bool success, ) = payable(msg.sender).call{value: fixedRewardAmount}("");
        require(success, "ETH transfer failed");

        emit RewardClaimed(_messageHash, msg.sender, recoveredSigner);
    }
```

Function Checks:

- **Unique Hash:** Prevents replay (checks `claimedHashes`).
- **Funds:** Contract balance sufficient?
- **Signature:** Valid proof from correct signer (Base/Device)? Uses `ECDSA.recover`.

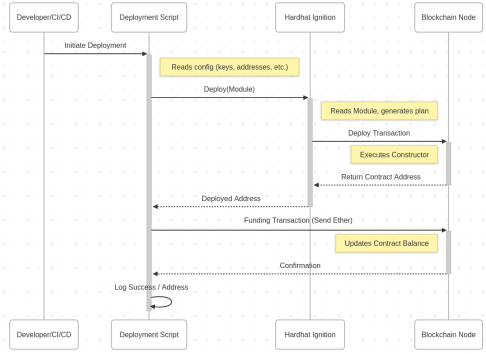
Actions (If Checks Pass):

- **Mark Claimed:** Updates `claimedHashes`.
- **Transfer Reward:** Sends funds to Intermediary (`msg.sender`).
- **Emit Event:** Logs `RewardClaimed`.

Analysis Approach & Implementation Scope

Implementation:

- Functional simulation using Node.js & Solidity (Hardhat).
- Goal: Validate core crypto and reward logic flow.



Scope:

- Security: Based on Threat Model (confidentiality, integrity, etc.).
- Performance: Estimates for Compute, Communication overhead, Gas costs.

Limitation:

- No empirical hardware/network tests conducted.

Analysis Results: Security & Performance

Security Analysis: Threat & Defense Summary

Threat	Primary Defense
Eavesdropping	AES-GCM
Tampering	AES-GCM Tag
Impersonation	ECDSA Signatures
Reward Replay	`claimedHashes` Mapping
MitM	ECDSA + E2E Crypto

Result: Theoretically robust against modeled threats.

Performance Analysis: Key Findings

- **Compute:** Dominated by Elliptic Curve Cryptography (ECC) operations (ECDH, ECDSA).
- **Communication:** Adds ∼152 bytes fixed overhead (keys, tag, signature); significant percentage increase for small IoT payloads.
- **Blockchain (Gas):** Primarily driven by State Writes (`SSTORE` for `claimedHashes`) and Signature Verification (`ecrecover` precompile).
- **Implication:** Provides strong security but incurs notable resource costs (compute, bandwidth, gas) for small messages.

Discussion & Future Work

Discussion: Interpretation & Trade-offs

Interpretation of Results:

- The proposed design offers strong guarantees for end-to-end confidentiality, integrity, and authenticity.
- The blockchain-based incentive mechanism is functionally sound, reliably rewarding intermediaries based on verifiable cryptographic proof.

The Core Trade-off:

- Achieving this high level of security and trust-minimized verification comes at a significant cost.
- **High Costs Incurred:**
  - Compute: Intensive ECC operations on devices/servers.
  - Communication: Increased packet size due to cryptographic overhead.
  - Blockchain Gas: State changes (`claimedHashes`) and signature checks (`ecrecover` precompile) on-chain are expensive.

Feasibility Considerations:

- **Practicality:** Deployment depends heavily on specific application constraints and network conditions.
- **Economic Viability:** System sustainability requires the `Reward Value > Gas Cost`.

Future Work: Validation, Optimization & Enhancements

Validation & Optimization (Near-Term Priorities):

- **Empirical Testing (High Priority):**
  - Measure real-world performance: hardware energy consumption, network latency.
  - Determine actual blockchain gas costs on testnets/mainnet.
- **Cost Reduction:**
  - Explore Layer 2 solutions or alternative blockchains to minimize gas fees.
  - Optimize cryptographic library implementations for efficiency.
- **Integration:**
  - Adapt the system to work alongside standard IoT protocols (e.g., wrappers for `MQTT`/`CoAP`).

Enhancements & Exploration (Longer-Term):

- **Advanced Contract Logic:**
  - Implement dynamic reward structures (e.g., based on data value, location).
  - Introduce reputation systems for Intermediaries or Devices.
- **Alternative Approaches:**
  - Investigate different cryptographic primitives (e.g., post-quantum crypto).
  - Explore blockchain scalability patterns (State Channels, transaction batching) if applicable to the use case.
- **Formal Verification:**
  - Apply formal methods to rigorously prove the security properties of the smart contract and protocol.

Conclusion

Conclusion & Contributions

- **Proposed & Analyzed:** An integrated system combining end-to-end security with a blockchain-based incentive mechanism for IoT data relay via trustless intermediaries.
- **Key Contributions:**
  - **Novel Architecture:** Defined the roles and interactions between Sender, Receiver, Intermediary, and Smart Contract.
  - **Protocol Specification:** Detailed the cryptographic steps for secure session establishment, data exchange, and reward proof generation.
  - **Smart Contract Design:** Implemented the `MessageReward` contract in Solidity for automated, trustless reward claim verification.
  - **Theoretical Analysis:** Evaluated the security properties and estimated performance (compute, communication, gas).
- **Main Takeaway:** While promising, *empirical validation* (hardware testing, real-world network deployment, actual gas cost measurement) is the essential next step to determine practical feasibility and optimize performance.

Thank You

Project Team:

Aninda Nath, Gaurav Kumar Sharma, Dwaipayan Biwas, Shibam Nath

Repository: `https://github.com/Aninda001/IoT-blockchain`