Secure Multi-Authority Hierarchical Access Control for Industrial IoT using ECC-Based CP-ABE

Abhradeep Das (2024202018)
Lakshay Baijal (2024202006)

M.Tech CSIS, IIIT Hyderabad

11th November, 2025

Introduction

- The **Industrial Internet of Things (IIoT)** connects thousands of devices and sensors that continuously share sensitive operational data.
- Ensuring **secure, fine-grained, and scalable access control** is a major challenge due to limited device resources.
- Traditional solutions:
  - Centralized access control $\rightarrow$ single point of failure.
  - Pairing-based CP-ABE $\rightarrow$ computationally expensive for embedded nodes.
- **Our approach:**
  - A lightweight **ECC-based Multi-Authority CP-ABE** system.
  - Integrates a hierarchical trust model (Root Authority $\rightarrow$ Sub-Authorities $\rightarrow$ AAs).
  - Employs an **Edge Authority (EA)** for partial decryption assistance without compromising privacy.
- Designed specifically for **resource-constrained IIoT deployments**.

Motivation

- **Challenge 1: Heavy Computation in IoT Devices**
  Most ABE systems rely on pairing-based cryptography, requiring high CPU cycles and memory — unsuitable for embedded systems.
- **Challenge 2: Centralized Trust Model**
  A single authority managing all attributes leads to bottlenecks, privacy risks, and poor scalability across organizations.
- **Challenge 3: Real-Time Decryption Demand**
  Industrial sensors need quick access verification without offloading full decryption load to cloud or central servers.
- **Our Motivation:**
  - Use **Elliptic Curve Cryptography (ECC)** for efficiency — smaller keys, faster scalar operations.
  - Employ **Multi-Authority (MA)** hierarchy to distribute attribute management.
  - Introduce **Edge-Assisted Decryption** to offload heavy computation securely.

Technology Stack & Tools

Programming Language and Environment:

- Implemented entirely in **Python 3.12**.
- Virtual environment managed using `venv`.

Core Cryptographic Libraries:

- `ECPy` — for Elliptic Curve Cryptography on `secp256r1`.
- `cryptography` — AES-GCM encryption and decryption.
- `hashlib` — SHA-256 for key derivation.

Mathematical Components:

- Shamir's Secret Sharing implemented for attribute-based threshold enforcement.
- Elliptic curve arithmetic for scalar multiplication and point addition.

Data Handling & Analysis:

- `numpy`, `pandas`, and `matplotlib` for performance benchmarks.
- `networkx` for future access-structure visualization.

Execution Scripts:

- Automated demo: `one_click_multi_aa_upload_demo.sh`
- Performance test: `benchmarks/perf_benchmark.py`

Cryptosystem Overview: Motivation & Design Choice

Why ECC-based Cryptosystem?

- Traditional ABE schemes rely on **pairing-based cryptography** (e.g., bilinear maps).
- Pairings are computationally heavy for **IoT or edge devices**.
- **Elliptic Curve Cryptography (ECC)** offers equivalent security at smaller key sizes:
  - 256-bit ECC 3072-bit RSA in strength.
  - Fewer modular multiplications $\rightarrow$ faster and energy-efficient.

Why Multi-Authority Hierarchy?

- Prevents single-point trust failure.
- Each Sub-Authority manages distinct attributes (domain-wise, e.g., Finance, IT).
- Enables scalable and distributed key management.

Cryptographic Components Used:

1. ECC (`secp256r1`) — public key generation, ElGamal transform.
2. AES-GCM — fast symmetric encryption of data.
3. Shamir's Secret Sharing — enforces attribute-based access threshold.
4. SHA-256 — deterministic key derivation from EC points.

Cryptosystem Overview: ECC and Key Generation

Elliptic Curve Cryptography (ECC):

- Curve used: `secp256r1`.
- Group operation: point addition and scalar multiplication over finite field.
- Base point G acts as generator for key derivation.

Key Generation:

$$\text{Private key: } x \in_R [1, n-1], \quad \text{Public key: } P = xG$$

- Each authority (`AA`) and user generates ECC key pairs.
- Root Authority distributes trust by delegating x_i values to sub-authorities.

Advantages over RSA/Pairing Schemes:

- Smaller key sizes reduce computation time and bandwidth.
- Compatible with lightweight IoT processors.
- Secure under the Elliptic Curve Discrete Logarithm Problem (ECDLP).

Implementation:

- `ecpy.curves.Curve.get_curve('secp256r1')`
- Functions: `scalar_mul(k,P)`, `point_add(P,Q)`, `hash_to_int(data)`.

Cryptosystem Overview: AES-GCM and Key Derivation

Why AES-GCM?

- Provides both **confidentiality** and **integrity**.
- Authenticated encryption mode prevents tampering.
- GCM is faster and parallelizable compared to CBC or CFB modes.

Key Derivation from ECC:

- Instead of random AES keys, derive it deterministically from EC point sG.

$$K = \mathrm{SHA256}(\mathrm{point\_to\_bytes}(sG))$$

- Ensures both encryptor and decryptor obtain identical AES key using shared scalar s.

Encryption Process:

1. Choose random scalar s.
2. Derive AES key K from sG.
3. Encrypt plaintext using:

$$\mathrm{AESGCM}(K).\mathit{encrypt}(\mathrm{nonce}, \mathrm{plaintext})$$

4. Output $\{\mathrm{nonce}, \mathrm{ct}, \mathrm{P\_bytes\_b64}\}$.

Decryption:

- Compute same sG at receiver side.
- Re-derive $K = H(sG)$ and decrypt using AES-GCM.

Cryptosystem Overview: Shamir's Secret Sharing & ElGamal Transform

Shamir's Secret Sharing (SSS):

- Used to distribute scalar s among n attributes.
- Secret polynomial: $f(x) = s + a_1 x + a_2 x^2 + \cdots + a_{t-1} x^{t-1}$.
- Each attribute receives a share $(x_i, f(x_i))$.
- Threshold property: any t shares can reconstruct s.

Why Shamir's Scheme?

- Linear operations — easy to implement over $\mathbb{Z}_p$.
- Supports dynamic threshold adjustment (e.g., $t = 3$ of 5 attributes).

ElGamal-Based Edge Transformation:

- Edge Authority (EA) helps decrypt without learning s.

$$C_1 = rG, \quad C_2 = sG + rD_{\text{pub}}$$

- User derives:

$$sG = C_2 - dC_1$$

- This blinded transformation allows lightweight user decryption.

Security:

- EA never learns d or K.
- Resistant to replay and collusion due to threshold reconstruction.

Workflow 1: Multi-Authority Encryption

1. **Attribute Assignment:** Each attribute (e.g., `attrA`, `attrC`) is managed by its own Attribute Authority (AA).
2. **Secret Sharing:** A random scalar s is generated and divided into n shares using Shamir's (n, t) scheme.
3. **Share Distribution:** Each share is tagged with its corresponding attribute and owner AA.
4. **Ciphertext Generation:**
   - Compute $P = sG$ and derive AES key $K = H(P)$.
   - Encrypt data with AES-GCM $\rightarrow$ produces $(\text{nonce}, \text{ct})$.
5. **Vault Creation:** All attribute shares are stored in a secure `vault.json`.

Illustration: RA $\rightarrow$ AAs $\rightarrow$ Attributes $\rightarrow$ Ciphertext + Shares

Workflow 2: Edge-Assisted Decryption (EA Transform)

1. **EA Pre-Decrypt:**
   - EA verifies user's authorized attributes.
   - Fetches corresponding shares from `vault.json`.
   - Reconstructs or blinds s (depending on threshold policy).
2. **ElGamal-Based Transformation:**

   $$C_1 = rG, \quad C_2 = sG + rD_{\text{pub}}$$

   - EA sends (C_1, C_2) as a transform token to the user.
3. **User Final Decryption:**

   $$sG = C_2 - dC_1, \quad K = H(sG)$$

   - User derives K and decrypts AES-GCM ciphertext to recover original data.

Threat Model & Security Mechanisms

1. Collusion Attack:

- Unauthorized users may try to combine their shares.
- Prevented by Shamir's threshold t: fewer than t shares reveal nothing.

2. Malicious Edge Authority:

- EA cannot recover AES key since it lacks user secret d.
- ElGamal transformation ensures s remains hidden.

3. Key Exposure Attack:

- Even if transformed keys (TSKs) are leaked, they depend on d.

4. Integrity & Authenticity:

- AES-GCM provides built-in authentication.
- Any ciphertext tampering causes decryption failure.

Performance & Results

Benchmark Setup:

- Measured runtime for ECC scalar multiplication and AES-GCM encryption.
- System: Intel i7 CPU, Python 3.12, ECPy (secp256r1).

Results:

- ECC operations scale linearly with the number of keys (up to 1000 ops).
- AES-GCM overhead is minimal ($<$2 ms for 1 MB files).
- Our ECC-based scheme reduces total computation by 60–70% vs pairing-based CP-ABE.

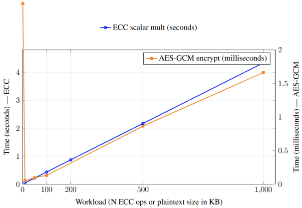

Figure: ECC vs AES-GCM runtime on increasing workload.

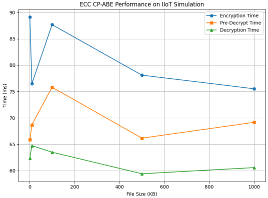

Figure: ECC CP-ABE Performance on IIoT Simulation — Measured encryption, edge pre-decr

Conclusion & Future Work

Summary:

- Implemented an **ECC-based Multi-Authority Hierarchical CP-ABE** system.
- Enabled lightweight access control for Industrial IoT.
- Edge-assisted decryption offloads computation from constrained devices.

Future Research Directions:

- Efficient **attribute revocation** and key updates.
- Integration with real IoT hardware (e.g., Raspberry Pi).
- Lattice-based or post-quantum cryptography for resilience.
- Formal proofs of security under standard assumptions.

> "Secure, Scalable, and Lightweight Access Control for the Future of IIoT"

References I

[1] Bethencourt, J., Sahai, A., Waters, B. (2007). Ciphertext-Policy Attribute-Based Encryption.

[2] Chase, M. (2007). Multi-Authority ABE.

[3] Ruj, S., Nayak, A., Stojmenovic, I. (2013). Decentralized Access Control in Cloud Using ABE.

Thank You!

Questions?

Contact: `lakshaybaijal@gmail.com`
`abhradeepdas10@gmail.com`