📋 TASK 2: IMPLEMENT SYMMETRIC ENCRYPTION

Priority: CRITICAL
Estimated Time: 3-4 days
Status: NOT STARTED
Dependencies: TASK 1 must be completed first (need authentication keys)

🎯 WHAT THIS TASK IS ABOUT

The Requirement:

> "Encryption of data between the Central and the CPs using symmetric encryption. The encryption keys do not reside in the code. All messages between the CPs and Central will be encrypted."

In Simple Words:

Right now ALL messages between CP and Central are sent in plain text. Anyone sniffing the network can read:

- "SUPPLY_UPDATE#CP-001#5.2#1.56" ← Anyone can see this!

After this task:

- "a8f3kd92jf0w..." ← Encrypted, unreadable!

How it works:

1. After authentication, Central gives each CP a unique encryption key
2. CP uses that key to encrypt every message
3. Central uses same key to decrypt
4. Each CP has a DIFFERENT key

❌ WHAT'S WRONG NOW

```
CP_Engine → Central: "SUPPLY_UPDATE#CP-001#5.2#1.56"
                ↑
                Plain text! Anyone can read!

Central → CP_Engine: "AUTHORIZE#DRIVER-001#CP-001#10"
```

```
                ↑
Plain text! Anyone can modify!
```

Problems:

- Zero encryption
- Man-in-the-middle attacks possible
- Anyone can fake messages
- Will fail correction immediately

✅ WHAT IT SHOULD BE

```
CP_Engine has: encryption_key = "abc123xyz..."
                ↓
CP_Engine: Encrypt "SUPPLY_UPDATE#CP-001#5.2#1.56"
                ↓
CP_Engine → Central: "gAAAAABl3k8fQ2..."  (encrypted blob)
                ↓
Central: Decrypt with same key
                ↓
Central gets: "SUPPLY_UPDATE#CP-001#5.2#1.56"

Same in reverse direction!
```

🔧 WHAT YOU NEED TO DO

Step 1: Choose Encryption Library

Recommended: Python Cryptography (Fernet)

- Simple symmetric encryption
- Already in Python
- Perfect for this use case

How Fernet works:

```python
from cryptography.fernet import Fernet

# Generate key (Central does this after authentication)
key = Fernet.generate_key()  # Returns bytes like b'abc123...'

# Encrypt (CP does this before sending)
cipher = Fernet(key)
encrypted = cipher.encrypt(b"SUPPLY_UPDATE#CP-001#5.2#1.56")

# Decrypt (Central does this after receiving)
decrypted = cipher.decrypt(encrypted)  # Returns original message
```

Step 2: Create Encryption Helper Module

Create new file: `shared/encryption.py`

What to put in it:

- Function to generate encryption key
- Function to encrypt message
- Function to decrypt message
- Error handling for bad keys
- Comments explaining each function

Purpose:

- Both CP and Central will use this
- Keeps encryption code in one place
- Easy to test independently

Step 3: Modify Central to Generate Keys

File: `central/ev_central.py`

What to add:

In `_handle_authenticate()` method:

- After Registry confirms credentials are valid
- Generate unique encryption key for this CP
- Store it: `self.cp_encryption_keys[cp_id] = key`
- Send key to CP in `AUTHENTICATED` response
- Format: `AUTHENTICATED#CP-001#BASE64_ENCODED_KEY`

In message receiving:

- Before processing ANY message from a CP
- Check if CP is authenticated
- Get CP's encryption key
- Decrypt the message
- Then process as normal

In message sending:

- Before sending to CP
- Get CP's encryption key
- Encrypt the message
- Send encrypted version

Step 4: Modify CP_Engine to Use Encryption

File: `charging_point/ev_cp_engine.py`

What to add:

Store encryption key:

- After CP_Monitor authenticates successfully
- CP_Monitor passes key to CP_Engine
- CP_Engine stores it: `self.encryption_key = key`

Before sending ANY message to Central:

- Encrypt it using the key
- Send encrypted version
- Even STX/ETX/LRC protocol still works (you encrypt the DATA part)

After receiving ANY message from Central:

- Decrypt it first
- Then parse as normal

Step 5: Update Protocol to Handle Encryption

File: `shared/protocol.py`

What to think about:

Option A: Encrypt the whole frame

```
Original: STX + "SUPPLY_UPDATE#..." + ETX + LRC
Encrypted: Encrypt entire thing
```

Problem: LRC check won't work anymore

Option B: Encrypt only DATA (RECOMMENDED)

```
Original DATA: "SUPPLY_UPDATE#CP-001#5.2#1.56"
Encrypted DATA: "gAAAAABl3k8fQ2..."
Frame: STX + encrypted_data + ETX + LRC
```

Better: Frame structure preserved, only content encrypted

🧪 HOW TO TEST

Test 1: Generate Keys

```
python
```

```python
# Test script
from cryptography.fernet import Fernet

# Generate key
key = Fernet.generate_key()
print(f"Generated key: {key}")

# Create cipher
cipher = Fernet(key)

# Test message
message = b"SUPPLY_UPDATE#CP-001#5.2#1.56"
print(f"Original: {message}")

# Encrypt
encrypted = cipher.encrypt(message)
print(f"Encrypted: {encrypted}")

# Decrypt
decrypted = cipher.decrypt(encrypted)
print(f"Decrypted: {decrypted}")

# Should match original!
assert message == decrypted
print("✅ Encryption/Decryption works!")
```

Expected: Message encrypted and decrypted successfully

Test 2: Different Keys

```python
```

```
# Generate two different keys
key1 = Fernet.generate_key()
key2 = Fernet.generate_key()

cipher1 = Fernet(key1)
cipher2 = Fernet(key2)

# Encrypt with key1
encrypted = cipher1.encrypt(b"TEST")

# Try decrypt with key2 (SHOULD FAIL!)
try:
    cipher2.decrypt(encrypted)
    print("❌ BUG: Different key worked!")
except:
    print("✅ Different key correctly rejected!")
```

Expected: Can't decrypt with wrong key

Test 3: End-to-End with CP

```
1. Start system
2. Create CP using CP Manager
3. Authentication happens → CP gets encryption key
4. Check CP_Engine logs:
   - Should see: "Received encryption key: xxx"
   - Should see: "Encrypting message before send"
5. Check Central logs:
   - Should see: "Decrypting message from CP-001"
   - Should see: "Decrypted: SUPPLY_UPDATE#CP-001#..."
6. Start a charge
7. Watch encrypted messages in logs
8. Verify charging still works normally
```

Expected: Everything works, but messages are encrypted

Test 4: Key Mismatch

```
1. Manually modify CP_Engine code to use wrong key
```

```
2. Try to send message to Central
3. Central should:
   - Fail to decrypt
   - Log error: "Decryption failed for CP-001"
   - Reject the message
   - Maybe kick out the CP
```

Expected: Bad encryption detected and handled

📋 STEP-BY-STEP IMPLEMENTATION

Day 1: Create Encryption Module

Hour 1-2: Set Up

- Create `shared/encryption.py`
- Import Fernet
- Write generate_key() function
- Write encrypt_message() function
- Write decrypt_message() function

Hour 3-4: Test Module

- Create test script
- Test key generation
- Test encryption/decryption
- Test wrong key rejection
- Make sure it all works before integrating

Hour 5-6: Documentation

- Add comments to all functions
- Write examples
- Test edge cases (empty messages, special characters)

Day 2: Integrate with Central

Hour 1-2: Key Generation

- Modify `_handle_authenticate()`
- Generate key after validation
- Store in `self.cp_encryption_keys{}`
- Send key in AUTHENTICATED response

Hour 3-4: Decrypt Incoming

- Modify `_process_message()`
- Before parsing, decrypt the message
- Handle decryption errors
- Log encrypted vs decrypted messages

Hour 5-6: Encrypt Outgoing

- Before sending to CP, encrypt
- Test with AUTHORIZE messages
- Test with DENY messages
- Make sure all CP↔Central messages encrypted

Day 3: Integrate with CP_Engine

Hour 1-2: Receive Key

- Modify CP_Monitor to pass key to Engine
- Engine stores key in self.encryption_key
- Verify key is received correctly

Hour 3-4: Encrypt Outgoing

- Before sending to Central, encrypt
- Test SUPPLY_UPDATE messages
- Test SUPPLY_END messages

- Test HEARTBEAT messages

Hour 5-6: Decrypt Incoming

- Decrypt messages from Central
- Test AUTHORIZE messages
- Test STOP/RESUME commands
- Make sure everything still works

Day 4: Testing & Debugging

Hour 1-3: Full System Test

- Start complete system
- Create multiple CPs
- Each should get different key
- Test charging at each CP
- Watch logs for encrypted messages

Hour 4-6: Edge Cases

- Test key revocation (preparation for TASK 5)
- Test wrong key handling
- Test Central restart (keys lost, CPs re-authenticate)
- Test CP restart (gets same key from Registry)
- Fix any bugs

📋 CHECKLIST

Before marking complete:

- ☐ Created shared/encryption.py module
- ☐ Tested encryption/decryption independently
- ☐ Central generates unique key per CP

- [ ] Central stores keys in memory
- [ ] Central encrypts all outgoing messages to CPs
- [ ] Central decrypts all incoming messages from CPs
- [ ] CP_Engine receives key from Monitor
- [ ] CP_Engine encrypts all outgoing messages
- [ ] CP_Engine decrypts all incoming messages
- [ ] Each CP has different encryption key
- [ ] Wrong key is detected and rejected
- [ ] Tested with multiple CPs simultaneously
- [ ] Charging still works with encryption
- [ ] Logs show encrypted vs decrypted messages
- [ ] Code is clean and commented
- [ ] Ready to explain to teacher

💡 WHAT TO TELL TEACHER

Teacher asks: "Explain the encryption mechanism"

Your answer: "We implement symmetric encryption using the Fernet algorithm from Python's cryptography library. After successful authentication, Central generates a unique encryption key for each charging point using cryptographically secure random generation. This key is transmitted once during the authentication handshake and stored in memory on both sides. All subsequent messages between the CP Engine and Central - including supply updates, authorization commands, and status messages - are encrypted with this key before transmission. The STX/ETX/LRC frame structure is preserved, but the data payload itself is encrypted. Each CP has a unique key, so compromising one CP doesn't affect others. Keys are stored in memory only, never hardcoded, and are regenerated on re-authentication."

⚠️ COMMON PROBLEMS

Problem 1: "Encryption works but messages not received"

- **Fix:** Make sure you're encrypting the DATA part, not the STX/ETX/LRC
- The framing protocol needs those bytes as-is

Problem 2: "Keys getting corrupted"

- **Fix:** Fernet keys are BYTES, not strings
- Don't convert to string unless you base64 encode first

- When sending over network, base64 encode the key

Problem 3: "Different keys every time I restart"

- **Fix:** That's actually correct! Keys are in-memory only
- CP must re-authenticate after Central restarts
- This is security feature, not bug

Problem 4: "Can't decrypt - InvalidToken error"

- **Fix:** CP and Central using different keys
- Check authentication properly passed key to Engine
- Verify you're using same key for encrypt/decrypt

✅ SUCCESS CRITERIA

This task is complete when:

1. ✅ All messages between CP and Central are encrypted
2. ✅ Each CP has unique encryption key
3. ✅ Keys are generated dynamically (not hardcoded)
4. ✅ Wrong keys are detected and rejected
5. ✅ Charging works normally with encryption
6. ✅ Can see encrypted messages in logs
7. ✅ Teacher can verify encryption is happening

Status: Not started
Next Task: TASK 3 (Audit Log)