📋 TASK 4: ADD HTTPS TO REGISTRY

Priority: HIGH
Estimated Time: 1 day
Status: NOT STARTED
Dependencies: None (can do in parallel)

🎯 WHAT THIS TASK IS ABOUT

The Requirement:

> "Communication between EV_Registry and EV_CP_M must be carried out using the Establishment of a secure channel (HTTPS, SSL, RSA, …)."

In Simple Words:

Right now, Registry API uses HTTP. When CP sends credentials, anyone can intercept them!

Current:

```
CP_Monitor → Registry: "POST http://registry:5001/register"
                ↑
                HTTP = Not secure!
                Anyone can see username/password!
```

After this task:

```
CP_Monitor → Registry: "POST https://registry:5001/register"
                ↑
                HTTPS = Encrypted channel
                SSL protects credentials in transit
```

❌ WHAT'S WRONG NOW

Current setup:

- Registry runs on port 5001 with HTTP
- CP_Monitor connects with HTTP

- Credentials sent in plain text (even though inside JSON)
- Man-in-the-middle can steal credentials

Problems:

- Not secure
- Correction guide explicitly requires HTTPS
- Will lose points

✅ WHAT IT SHOULD BE

New setup:

- Registry runs on port 5001 with HTTPS
- Has SSL certificate (self-signed OK for lab)
- CP_Monitor connects with HTTPS
- Credentials encrypted during transport
- Still vulnerable to MITM with self-signed cert, but better than nothing

---

🔧 WHAT YOU NEED TO DO

Step 1: Generate Self-Signed SSL Certificate

Why self-signed:

- You don't have a real domain
- Don't need to pay for certificate
- Good enough for lab environment
- Teacher will accept this

How to generate:

Option A: Using OpenSSL (if installed):

```bash
```

```
openssl req -x509 -newkey rsa:4096 -nodes -out registry_cert.pem -keyout registry_key.pem -days 365
```

Option B: Using Python (easier): Create script `generate_cert.py`:

```
python
```

```python
from cryptography import x509
from cryptography.x509.oid import NameOID
from cryptography.hazmat.primitives import hashes
from cryptography.hazmat.primitives.asymmetric import rsa
from cryptography.hazmat.primitives import serialization
import datetime

# Generate private key
private_key = rsa.generate_private_key(
    public_exponent=65537,
    key_size=2048
)

# Generate certificate
subject = issuer = x509.Name([
    x509.NameAttribute(NameOID.COUNTRY_NAME, "ES"),
    x509.NameAttribute(NameOID.STATE_OR_PROVINCE_NAME, "Valencia"),
    x509.NameAttribute(NameOID.LOCALITY_NAME, "Alicante"),
    x509.NameAttribute(NameOID.ORGANIZATION_NAME, "EV Charging"),
    x509.NameAttribute(NameOID.COMMON_NAME, "registry"),
])

cert = x509.CertificateBuilder().subject_name(
    subject
).issuer_name(
    issuer
).public_key(
    private_key.public_key()
).serial_number(
    x509.random_serial_number()
).not_valid_before(
    datetime.datetime.utcnow()
).not_valid_after(
    datetime.datetime.utcnow() + datetime.timedelta(days=365)
).sign(private_key, hashes.SHA256())

# Write certificate
with open("registry/registry_cert.pem", "wb") as f:
    f.write(cert.public_bytes(serialization.Encoding.PEM))

# Write private key
with open("registry/registry_key.pem", "wb") as f:
    f.write(private_key.private_bytes(
        encoding=serialization.Encoding.PEM,
        format=serialization.PrivateFormat.TraditionalOpenSSL,
        encryption_algorithm=serialization.NoEncryption()
    ))

print("✅ Certificate generated!")
```

Run it: `python generate_cert.py`

Result:

- `registry/registry_cert.pem` (certificate)
- `registry/registry_key.pem` (private key)

Step 2: Modify Registry to Use HTTPS

File: `registry/ev_registry.py`

Import SSL module:

```python
import ssl
```

At the end of file, change:

Before (HTTP):

```python
if __name__ == "__main__":
    app.run(host='0.0.0.0', port=5001, debug=False)
```

After (HTTPS):

```python
```

```
if __name__ == "__main__":
    # Create SSL context
    ssl_context = ssl.SSLContext(ssl.PROTOCOL_TLS_SERVER)
    ssl_context.load_cert_chain(
        certfile='registry/registry_cert.pem',
        keyfile='registry/registry_key.pem'
    )

    print("[EV_Registry] Starting with HTTPS (SSL) on port 5001...")
    app.run(
        host='0.0.0.0',
        port=5001,
        debug=False,
        ssl_context=ssl_context
    )
```

Step 3: Update Dockerfile to Include Certificates

File: `Dockerfile.registry`

Add before CMD:

dockerfile

```dockerfile
# Copy SSL certificates
COPY registry/registry_cert.pem registry/
COPY registry/registry_key.pem registry/
```

Step 4: Modify CP_Monitor to Use HTTPS

File: `charging_point/ev_cp_monitor.py`

Where it calls Registry:

Before (HTTP):

python

```
response = requests.post(
    f"http://registry:5001/register",
    json={...}
)
```

After (HTTPS with verify=False for self-signed):

```python
response = requests.post(
    f"https://registry:5001/register",
    json={...},
    verify=False  # Ignore self-signed certificate warning
)
```

Add warning suppression at top:

```python
import urllib3
urllib3.disable_warnings(urllib3.exceptions.InsecureRequestWarning)
```

Note: In production, you'd verify the certificate. For lab, it's OK to skip.

Step 5: Update Central to Use HTTPS for Registry

File: `central/ev_central.py`

Wherever it calls Registry:

Change:

```python
response = requests.post(
    f"{REGISTRY_URL}/verify",
    json={...}
)
```

To:

```python
```

```
response = requests.post(
    f"{REGISTRY_URL}/verify",
    json={...},
    verify=False  # Self-signed cert
)
```

Update config:

File: `config.py`

Change:

```python
REGISTRY_URL = os.getenv("REGISTRY_URL", "http://localhost:5001")
```

To:

```python
REGISTRY_URL = os.getenv("REGISTRY_URL", "https://localhost:5001")
```

🧪 HOW TO TEST

Test 1: Certificate Generated

```bash
ls -la registry/
# Should see:
#   registry_cert.pem
#   registry_key.pem
```

Expected: Both files exist

Test 2: Registry Starts with HTTPS

```bash
```

```
docker-compose up registry

# Watch logs, should see:
# "[EV_Registry] Starting with HTTPS (SSL) on port 5001..."
# No SSL errors
```

Expected: Starts successfully

Test 3: Test with Browser

```
1. Open browser
2. Go to: https://localhost:5001/health
3. Browser shows security warning (self-signed cert)
4. Click "Advanced" → "Accept Risk"
5. Should see: {"status": "ok", "service": "EV_Registry"}
```

Expected: API responds via HTTPS

Test 4: Test with Postman

```
1. Open Postman
2. POST to https://localhost:5001/register
3. Turn off "SSL certificate verification" in settings
4. Body: {"cp_id": "CP-999", ...}
5. Should get: {"username": "...", "password": "..."}
```

Expected: API works via HTTPS

Test 5: CP_Monitor Connects via HTTPS

```
1. Start full system
2. Create CP using CP Manager
3. Watch CP_Monitor logs:
   - Should see: "Connecting to Registry at https://registry:5001"
   - Should NOT see SSL errors
   - Should see: "Got credentials: ..."
```

Expected: CP connects successfully via HTTPS

Test 6: Central Verifies via HTTPS

```
1. When CP authenticates
2. Watch Central logs:
  - Should see: "Verifying credentials with Registry (HTTPS)"
  - Should NOT see SSL errors
  - Should see: "Credentials valid"
```

Expected: Central verifies via HTTPS

📋 STEP-BY-STEP IMPLEMENTATION

Hour 1: Generate Certificate

- Create `generate_cert.py` script
- Run it
- Verify files created
- Test certificate is valid
- Copy to registry/ folder

Hour 2: Modify Registry

- Update `ev_registry.py`
- Add SSL context
- Test registry starts
- Test with browser

Hour 3: Update CP_Monitor

- Change HTTP to HTTPS
- Add verify=False
- Test connection works

- Handle SSL warnings

Hour 4: Update Central

- Change HTTP to HTTPS
- Update config.py
- Test verification works
- Handle SSL warnings

Hour 5-6: Integration Testing

- Start full system
- Test CP registration via HTTPS
- Test authentication via HTTPS
- Verify logs show HTTPS connections
- Fix any issues

Hour 7-8: Documentation

- Document why self-signed
- Add comments to code
- Prepare explanation for teacher
- Test final deployment

📋 CHECKLIST

Before marking complete:

- [ ] Generated SSL certificate and key
- [ ] Files in registry/registry_cert.pem and registry_key.pem
- [ ] Registry starts with HTTPS
- [ ] Browser can access via HTTPS (with warning)
- [ ] CP_Monitor connects via HTTPS
- [ ] Central verifies via HTTPS
- [ ] No SSL errors in logs
- [ ] verify=False used for self-signed cert

- [ ] config.py updated to HTTPS
- [ ] Dockerfile.registry includes certificates
- [ ] Tested full authentication flow
- [ ] Ready to explain to teacher

💡 WHAT TO TELL TEACHER

Teacher asks: "Why self-signed certificate?"

Your answer: "We use a self-signed certificate because this is a laboratory environment without a registered domain name. In production, we would use a certificate from a trusted Certificate Authority like Let's Encrypt. The self-signed certificate still provides encryption of data in transit between the CP Monitor and Registry, protecting credentials during the authentication handshake. We disable certificate verification in the client code (verify=False) because Python's requests library won't trust our self-signed certificate by default. This is acceptable for lab demonstration but would not be used in production."

Teacher asks: "Show me HTTPS is working"

Your action:

1. Open browser
2. Navigate to https://localhost:5001/health
3. Show security warning (proves HTTPS)
4. Accept and show API response
5. Open CP_Monitor logs
6. Point to "https://registry:5001" in connection string

⚠️ COMMON PROBLEMS

Problem 1: "SSL module not found"

- **Fix:** Install: `pip install pyopenssl`

Problem 2: "Certificate file not found"

- **Fix:** Make sure path is correct relative to where python runs
- Use absolute path or copy to correct location

Problem 3: "SSL handshake failed"

- **Fix:** Make sure both cert AND key files are loaded
- Check file permissions (must be readable)

Problem 4: "Certificate has expired"

- **Fix:** Regenerate with longer validity
- In script, change `timedelta(days=365)` to `timedelta(days=3650)`

Problem 5: "Connection refused"

- **Fix:** Make sure registry is actually listening on HTTPS
- Check it's not still on HTTP

✅ SUCCESS CRITERIA

This task is complete when:

1. ✅ Registry runs on HTTPS (port 5001)
2. ✅ Self-signed certificate generated
3. ✅ CP_Monitor connects via HTTPS
4. ✅ Central verifies via HTTPS
5. ✅ Browser can access via HTTPS (with warning)
6. ✅ No SSL errors in production logs
7. ✅ Can explain why self-signed and why verify=False

Status: Not started
Next Task: TASK 5 (Key Revocation)