```python
from fastapi import FastAPI, HTTPException
from pydantic import BaseModel, Field
from typing import List, Optional, Dict, Any
from datetime import datetime
import numpy as np
import hashlib
import uuid
import threading

app = FastAPI(title="Global Fusion Engine / Aggregator (Team 5)")

LOCK = threading.Lock()

# -------------------------
# Data Models
# -------------------------
class StateVector(BaseModel):
    x: float
    y: float
    z: float
    vx: float
    vy: float
    vz: float
    covariance: List[List[float]]

class RiskCell(BaseModel):
    cell_id: str
    risk_score: float

class OperatorSummary(BaseModel):
    operator_id: str
    timestamp: datetime
    objects: List[StateVector]
    grid_risks: List[RiskCell]
    signature: Optional[str] = None

class AggregatedObject(BaseModel):
    object_id: str
    mean_state: List[float]
    fused_covariance: List[List[float]]
    contributing_operators: List[str]

class CDMAlert(BaseModel):
    alert_id: str
    object_id: str
    collision_probability: float
    risk_level: str
    timestamp: datetime

# -------------------------
# Storage (in-memory)
# -------------------------
SUBMISSIONS: Dict[str, List[OperatorSummary]] = {}
FUSED_OBJECTS: Dict[str, AggregatedObject] = {}
ALERTS: List[CDMAlert] = []
GRID_RISK_MAP: Dict[str, float] = {}

# -------------------------
# Utility Functions
# -------------------------
def secure_aggregate(vectors: List[List[float]]) -> List[float]:
    arr = np.array(vectors)
    return np.mean(arr, axis=0).tolist()

def compute_collision_probability(covariance: List[List[float]]) -> float:
    det = np.linalg.det(np.array(covariance))
    prob = np.exp(-det % 10) / 100
    return float(prob)

def classify_risk(prob: float) -> str:
    if prob > 0.05:
        return "HIGH"
```

```
    elif prob > 0.01:
        return "MEDIUM"
    return "LOW"

# -------------------------
# API Endpoints
# -------------------------
@app.post("/submit_summary")
def submit_summary(summary: OperatorSummary):
    with LOCK:
        if summary.operator_id not in SUBMISSIONS:
            SUBMISSIONS[summary.operator_id] = []
        SUBMISSIONS[summary.operator_id].append(summary)
    return {"status": "received", "operator": summary.operator_id}

@app.post("/fuse_all")
def fuse_all():
    global FUSED_OBJECTS, ALERTS, GRID_RISK_MAP
    with LOCK:
        FUSED_OBJECTS = {}
        ALERTS = []
        GRID_RISK_MAP = {}

        all_objects: Dict[str, List[StateVector]] = {}
        for op, subs in SUBMISSIONS.items():
            for s in subs:
                for idx, obj in enumerate(s.objects):
                    obj_id = f"OBJ-{idx}"
                    if obj_id not in all_objects:
                        all_objects[obj_id] = []
                    all_objects[obj_id].append(obj)

                for r in s.grid_risks:
                    GRID_RISK_MAP[r.cell_id] = GRID_RISK_MAP.get(r.cell_id, 0) + r.risk_score

        for obj_id, states in all_objects.items():
            mean_state = secure_aggregate([
                [s.x, s.y, s.z, s.vx, s.vy, s.vz] for s in states
            ])
            covariances = np.mean([s.covariance for s in states], axis=0).tolist()

            fused = AggregatedObject(
                object_id=obj_id,
                mean_state=mean_state,
                fused_covariance=covariances,
                contributing_operators=list(SUBMISSIONS.keys())
            )
            FUSED_OBJECTS[obj_id] = fused

            prob = compute_collision_probability(covariances)
            risk = classify_risk(prob)
            alert = CDMAlert(
                alert_id=str(uuid.uuid4()),
                object_id=obj_id,
                collision_probability=prob,
                risk_level=risk,
                timestamp=datetime.utcnow()
            )
            ALERTS.append(alert)

    return {"status": "fusion_complete", "objects": len(FUSED_OBJECTS), "alerts": len(ALERTS)}

@app.get("/alerts")
def get_alerts() -> List[CDMAlert]:
    return ALERTS

@app.get("/grid_risk")
def get_grid_risk() -> Dict[str, float]:
    return GRID_RISK_MAP

@app.get("/fused_objects")
def get_fused_objects() -> List[AggregatedObject]:
```

```
        return list(FUSED_OBJECTS.values())
```