Problem Statement 17

WireGuard Fabric (Management–Peer, Token-Join)

Reference: IETF RFCs 768 (UDP), 7748 (X25519), 8439 (ChaCha20-Poly1305), 7693 (BLAKE2), 7519 (JWT for join tokens) over 8446 (TLS 1.3)/9113 (HTTP/2) for the control plane, 5389/8445 (STUN/ICE) for NAT traversal, and 1918/4193 (IPv4 private/IPv6 ULA) for overlay addressing in a distributed WireGuard Management–Peer fabric.

Objective

Design and implement a **distributed WireGuard VPN fabric** with a **Management–Peer architecture**. Management nodes coordinate the cluster, issue **join tokens**, and manage membership; **peer nodes** use tokens to join and form the worker/edge layer. The system must support **single-node** and **multi-management** clusters, **elastic peer scaling**, and **fault tolerance**.

Problem

Monolithic VPN setups don't scale or fail over cleanly. Build a fabric where:

1. **Management Layer**
    - Runs a management server (REST/gRPC).
    - Issues and revokes **time-bound join tokens** (scoped roles: mgmt/peer).
    - Maintains membership, IPAM (virtual IP pool), and per-node WireGuard configs.
    - Supports **N≥3 management nodes** with state replication (e.g., Postgres/etcd/raft) and leader election.
2. **Peer Layer**
    - Peers join using tokens, receive virtual IPs, and **auto-fetch** WG configs/keys.
    - Peers securely communicate with all other peers and management nodes.
    - Supports **horizontal scaling** to hundreds/thousands of peers.
3. **Setup Workflow (must demonstrate)**
    - **Step 1:** Start first management node (bootstrap cluster).
    - **Step 2:** Generate a **join token**.
    - **Step 3:** Use token to add another **management node** (cluster grows).
    - **Step 4:** Use token to add **peer nodes** at will.

Functional Requirements

- **Token lifecycle:** create, scope (mgmt/peer), TTL, single/multi-use, revoke.
- **IPAM:** allocate/reclaim virtual IPs; avoid collision; persist state.
- **Config delivery:** signed WG configs (pubkey, allowed-ips, endpoints); rotate keys without breaking tunnels.
- **Health & membership:** heartbeats, status API, graceful removal/quarantine.

- **Topology modes:**
    - **Single management node** (dev/test).
    - **Multi-management cluster** (redundancy/failover).
- **Security:** TLS for control-plane; per-node authZ; audit log of membership changes.

Non-Functional Requirements

- **Resilience:** loss of one management node must not disrupt existing tunnels.
- **Scalability:** show join rate and steady-state metrics for ≥100 peers.
- **Observability:** metrics (join latency, key-rotate time, churn), logs, and alerts.
- **Portability:** run across clouds/DC/edge (Docker/K8s or compose).

Deliverables

- **Management service** (containerized) with APIs & minimal UI/CLI.
- **Peer join agent** (script/binary) that exchanges token → retrieves config → brings up wg interface.
- **Docs/Runbook:** bootstrap, add mgmt/peer, rotate keys, revoke, teardown.
- **Demo scripts:** one-shot lab showing Steps 1–4 and failover test.

Test Scenarios (must pass)

1. **Tokened joins:** peers and a second management node join via issued tokens.
2. **Failover:** kill one management node—fabric continues; add a new peer successfully.
3. **Scale-out:** add 50–100 peers; verify unique IPs and full mesh (or hub-and-spoke) reachability.
4. **Rotation & revoke:** rotate a peer key without tunnel drop; revoke a token and block joins.

Evaluation Criteria

- **Architecture & Reliability (30%)**: clean separation of mgmt/peer, HA behavior, safe state.
  Functionality (25%): token lifecycle, IPAM, config delivery, health.
- **Security (15%)**: TLS, authZ scopes, auditability, secret handling.
- **Scalability & Observability (15%)**: metrics, logs, demonstrated scale.
- **Docs & UX (15%)**: clear README/runbooks, scripts, and a simple operator UX.