CANONICAL LIVING OVERVIEW GUIDE

IPsec, TLS VPN, Remote Access, and Tunnel Troubleshooting

A tunnel and remote-access guide covering IPsec architecture, IKEv2, ESP, SAs, traffic selectors, NAT traversal, certificates, routing, policy, TLS VPN, client posture, split tunneling, clustering, mobility, MTU, observability, packet troubleshooting, and hybrid post-quantum transition.

PERMANENT ID
MYTHOS-FG-SEC-0004

PUBLICATION CLASS
FIELD GUIDE

VERSION / STATUS
1.0.0-candidate / CANDIDATE

PERMANENT PUBLICATION IDENTITY

IPsec, TLS VPN, Remote Access, and Tunnel Troubleshooting

Document ID
MYTHOS-FG-SEC-0004

Version
1.0.0-candidate

Status
Candidate Focused Field Guide

Review date
2026-10-26

Publication position
This is a living field guide. It is designed for frequent revision while retaining a stable permanent identity. Candidate status means the content is ready for structured use and review but is not represented as an external certification or authority publication.

LEVEL	CLASS	CADENCE	EVIDENCE
L1-L4	FIELD GUIDE	QUARTERLY	SOURCE-BOUND
Foundational through frontier	Practical and educational	Interim updates as needed	Limitations remain visible

Reader orientation

Dimension	Engineering position
Primary domain	MYTHOS-SEC - Cybersecurity, Security Engineering & Cryptography
Secondary domain	MYTHOS-NET; MYTHOS-QTC; MYTHOS-CLD
Audience	Network security, firewall, VPN, identity, remote-access, cloud, and support engineers.
Prerequisites	TCP/IP, routing, NAT, firewalls, PKI, cryptography, and packet capture fundamentals.
Publisher / owner	Mythos Systems / Aaron Stovall
Public classification	Public technical guidance

Expected outcomes

Trace IKE, IPsec, TLS, authentication, policy, routing, NAT, and application traffic end to end.

Design site-to-site and remote-access services with explicit cryptography, identity, selectors, routes, and failure behavior.

Diagnose negotiation, authentication, child-SA, selector, NAT, MTU, asymmetric path, and client problems.

Operate clusters, mobility, posture, split tunneling, DNS, and service access safely.

Plan hybrid and PQ transition while accounting for message size, fragmentation, interoperability, and lifecycle.

How to use this guide

Read the guide as a progressive system rather than a disconnected encyclopedia. Foundational material establishes the mental model; practitioner sections convert it into repeatable implementation and operations; advanced sections expose architecture, scale, security, and failure behavior; frontier sections describe technologies whose evidence or maturity is still evolving.

> Suggested reading path
> New practitioners should follow the chapters in order and complete the laboratories. Experienced engineers may begin with the architecture, failure, validation, or frontier sections, then use the related-document map to deepen a specific topic.

Learning and assurance model

Level	Reader outcome
L1 - Foundational	Terminology, first principles, component roles, packet or state flow, and simple working examples.
L2 - Practitioner	Implementation, configuration, automation, testing, troubleshooting, and operational evidence.
L3 - Advanced	Architecture, scale, concurrency, resilience, threat models, performance, and complex migrations.
L4 - Frontier	Emerging systems, research directions, technical uncertainty, readiness gates, and adoption signals.

Contents

CHAPTER 01

Tunnel security architecture

Understand what a tunnel protects, where protection begins and ends, and what remains exposed.

Chapter operating position

Understand what a tunnel protects, where protection begins and ends, and what remains exposed.

1.1 Security policy and association model

Security policy and association model must be engineered as an observable mechanism rather than a product label or checklist item. Relate policy databases, peer authorization, IKE SAs, child SAs, directions, lifetimes, and selectors. The design should name its authoritative inputs, state transitions, dependencies, limits, security boundaries, operational owner, telemetry, failure behavior, and acceptance criteria.

This guide traces the subject through the peer reachability, IKE exchange, authentication, child SA, selector, route, NAT, encapsulation, endpoint, and application chain. A configured feature, established session, successful control-plane exchange, or green dashboard is not sufficient proof. Intended state must be reconciled with applied state, real traffic or transactions, negative tests, degraded behavior, recovery, and the resulting service or security outcome.

A production implementation starts with a minimal known-good baseline and explicit invariants. Introduce one dependency or capability at a time, preserve before-state evidence, test positive and prohibited behavior, inject realistic failures, measure convergence or recovery, and retain a tested rollback. Automation must stop safely when authority, freshness, identity, capacity, or observed outcomes are ambiguous.

Troubleshooting locates the first divergence from the expected state machine or end-to-end transaction. Inspect both directions and all administrative boundaries; account for caching, eventual consistency, stale identity, time, recursive dependencies, asymmetric paths, encapsulation, hardware offload, partial failure, and missing telemetry. Closure requires a causal explanation, reproducible evidence, validated restoration, residual limitations, and prevention or monitoring actions.

Engineering practices

- Define authority, ownership, scope, dependencies, limits, and acceptance criteria for security policy and association model.
- Model the expected state, trust boundaries, and transaction before implementation or troubleshooting.
- Validate intended configuration, active control state, applied enforcement or forwarding, and end-to-end outcome independently.
- Test normal, prohibited, degraded, failed, recovery, upgrade, and rollback conditions.
- Preserve topology, identity, policy, configuration, packet, telemetry, log, timing, and service evidence.
- Automate through typed data, deterministic gates, canaries, bounded authority, and reversible changes.

Failure patterns

- Treating security policy and association model as a vendor feature instead of an end-to-end engineering system.
- Relying on intended configuration or controller state without proving applied and operational behavior.
- Ignoring reverse paths, external dependencies, identity, time, resource limits, or partial failure.
- Changing several variables before preserving evidence and stating a falsifiable hypothesis.
- Allowing automation or AI to act on stale, conflicting, unverified, or unauthorized inputs.
- Declaring success after one positive probe without negative tests, scale checks, rollback, and recurrence monitoring.

Validation and evidence

Method	Expected evidence
Examine	Requirements, architecture, identities, models, policy, configuration, ownership, and dependencies for security policy and association model.
Observe	Control state, applied policy or forwarding, telemetry, logs, packets, sessions, resources, and endpoint behavior.
Test	Expected behavior, prohibited behavior, dependency loss, partial failure, scale, recovery, and rollback.
Measure	Availability, latency, loss, convergence, risk, resource use, change success, detection quality, and user or mission outcome.
Reconcile	Intended state against observed network, security, identity, application, data, and evidence records.

1.2 Tunnel and transport modes

Tunnel and transport modes must be engineered as an observable mechanism rather than a product label or checklist item. Compare protected payloads, inner and outer headers, gateways, hosts, routing, visibility, and use cases. The design should name its authoritative inputs, state transitions, dependencies, limits, security boundaries, operational owner, telemetry, failure behavior, and acceptance criteria.

This guide traces the subject through the peer reachability, IKE exchange, authentication, child SA, selector, route, NAT, encapsulation, endpoint, and application chain. A configured feature, established session, successful control-plane exchange, or green dashboard is not sufficient proof. Intended state must be reconciled with applied state, real traffic or transactions, negative tests, degraded behavior, recovery, and the resulting service or security outcome.

A production implementation starts with a minimal known-good baseline and explicit invariants. Introduce one dependency or capability at a time, preserve before-state evidence, test positive and prohibited behavior, inject realistic failures, measure convergence or recovery, and retain a tested rollback. Automation must stop safely when authority, freshness, identity, capacity, or observed outcomes are ambiguous.

Troubleshooting locates the first divergence from the expected state machine or end-to-end transaction. Inspect both directions and all administrative boundaries; account for caching, eventual consistency, stale identity, time, recursive dependencies, asymmetric paths, encapsulation, hardware offload, partial failure, and missing telemetry. Closure requires a causal explanation, reproducible evidence, validated restoration, residual limitations, and prevention or monitoring actions.

Engineering practices

Define authority, ownership, scope, dependencies, limits, and acceptance criteria for tunnel and transport modes.

Model the expected state, trust boundaries, and transaction before implementation or troubleshooting.

Validate intended configuration, active control state, applied enforcement or forwarding, and end-to-end outcome independently.

Test normal, prohibited, degraded, failed, recovery, upgrade, and rollback conditions.

Preserve topology, identity, policy, configuration, packet, telemetry, log, timing, and service evidence.

Automate through typed data, deterministic gates, canaries, bounded authority, and reversible changes.

Implementation sketch

```
ikev2_profile:
  authentication: certificate
  proposals:
    encryption: aes-gcm-256
    prf: hmac-sha2-384
    key_exchange: ecp384
  transition:
    multiple_key_exchange: evaluate
    ppk: controlled-pilot
    fragmentation: enabled
  child_sa:
    local: 10.10.0.0/16
    remote: 10.20.0.0/16
  validation:
    - ike_auth_identity
    - selectors_and_routes
    - nat_and_mtu
    - rekey_and_failover
```

Failure patterns

Treating tunnel and transport modes as a vendor feature instead of an end-to-end engineering system.

Relying on intended configuration or controller state without proving applied and operational behavior.

Ignoring reverse paths, external dependencies, identity, time, resource limits, or partial failure.

Changing several variables before preserving evidence and stating a falsifiable hypothesis.

Allowing automation or AI to act on stale, conflicting, unverified, or unauthorized inputs.

Declaring success after one positive probe without negative tests, scale checks, rollback, and recurrence monitoring.

Validation and evidence

Method	Expected evidence
Examine	Requirements, architecture, identities, models, policy, configuration, ownership, and dependencies for tunnel and transport modes.
Observe	Control state, applied policy or forwarding, telemetry, logs, packets, sessions, resources, and endpoint behavior.
Test	Expected behavior, prohibited behavior, dependency loss, partial failure, scale, recovery, and rollback.
Measure	Availability, latency, loss, convergence, risk, resource use, change success, detection quality, and user or mission outcome.
Reconcile	Intended state against observed network, security, identity, application, data, and evidence records.

1.3 ESP, integrity, confidentiality, and replay

ESP, integrity, confidentiality, and replay must be engineered as an observable mechanism rather than a product label or checklist item. Understand authenticated encryption, sequence, anti-replay windows, negotiation, and overhead. The design should name its authoritative inputs, state transitions, dependencies, limits, security boundaries, operational owner, telemetry, failure behavior, and acceptance criteria.

This guide traces the subject through the peer reachability, IKE exchange, authentication, child SA, selector, route, NAT, encapsulation, endpoint, and application chain. A configured feature, established session, successful control-plane exchange, or green dashboard is not sufficient proof. Intended state must be reconciled with applied state, real traffic or transactions, negative tests, degraded behavior, recovery, and the resulting service or security outcome.

A production implementation starts with a minimal known-good baseline and explicit invariants. Introduce one dependency or capability at a time, preserve before-state evidence, test positive and prohibited behavior, inject realistic failures, measure convergence or recovery, and retain a tested rollback. Automation must stop safely when authority, freshness, identity, capacity, or observed outcomes are ambiguous.

Troubleshooting locates the first divergence from the expected state machine or end-to-end transaction. Inspect both directions and all administrative boundaries; account for caching, eventual consistency, stale identity, time, recursive dependencies,

asymmetric paths, encapsulation, hardware offload, partial failure, and missing telemetry. Closure requires a causal explanation, reproducible evidence, validated restoration, residual limitations, and prevention or monitoring actions.

Engineering practices

Define authority, ownership, scope, dependencies, limits, and acceptance criteria for esp, integrity, confidentiality, and replay.

Model the expected state, trust boundaries, and transaction before implementation or troubleshooting.

Validate intended configuration, active control state, applied enforcement or forwarding, and end-to-end outcome independently.

Test normal, prohibited, degraded, failed, recovery, upgrade, and rollback conditions.

Preserve topology, identity, policy, configuration, packet, telemetry, log, timing, and service evidence.

Automate through typed data, deterministic gates, canaries, bounded authority, and reversible changes.

Failure patterns

Treating esp, integrity, confidentiality, and replay as a vendor feature instead of an end-to-end engineering system.

Relying on intended configuration or controller state without proving applied and operational behavior.

Ignoring reverse paths, external dependencies, identity, time, resource limits, or partial failure.

Changing several variables before preserving evidence and stating a falsifiable hypothesis.

Allowing automation or AI to act on stale, conflicting, unverified, or unauthorized inputs.

Declaring success after one positive probe without negative tests, scale checks, rollback, and recurrence monitoring.

Validation and evidence

Method	Expected evidence
Examine	Requirements, architecture, identities, models, policy, configuration, ownership, and dependencies for esp, integrity, confidentiality, and replay.
Observe	Control state, applied policy or forwarding, telemetry, logs, packets, sessions, resources, and endpoint behavior.
Test	Expected behavior, prohibited behavior, dependency loss, partial failure, scale, recovery, and rollback.
Measure	Availability, latency, loss, convergence, risk, resource use, change success, detection quality, and user or mission outcome.
Reconcile	Intended state against observed network, security, identity, application, data, and evidence records.

1.4 Trust boundaries and residual exposure

Trust boundaries and residual exposure must be engineered as an observable mechanism rather than a product label or checklist item. Identify metadata, endpoints, administrators, routes, DNS, clients, applications, and management outside tunnel protection. The design should name its authoritative inputs, state transitions, dependencies, limits, security boundaries, operational owner, telemetry, failure behavior, and acceptance criteria.

This guide traces the subject through the peer reachability, IKE exchange, authentication, child SA, selector, route, NAT, encapsulation, endpoint, and application chain. A configured feature, established session, successful control-plane exchange, or green dashboard is not sufficient proof. Intended state must be reconciled with applied state, real traffic or transactions, negative tests, degraded behavior, recovery, and the resulting service or security outcome.

A production implementation starts with a minimal known-good baseline and explicit invariants. Introduce one dependency or capability at a time, preserve before-state evidence, test positive and prohibited behavior, inject realistic failures, measure convergence or recovery, and retain a tested rollback. Automation must stop safely when authority, freshness, identity, capacity, or observed outcomes are ambiguous.

Troubleshooting locates the first divergence from the expected state machine or end-to-end transaction. Inspect both directions and all administrative boundaries; account for caching, eventual consistency, stale identity, time, recursive dependencies, asymmetric paths, encapsulation, hardware offload, partial failure, and missing telemetry. Closure requires a causal explanation, reproducible evidence, validated restoration, residual limitations, and prevention or monitoring actions.

Engineering practices

Define authority, ownership, scope, dependencies, limits, and acceptance criteria for trust boundaries and residual exposure.

Model the expected state, trust boundaries, and transaction before implementation or troubleshooting.

Validate intended configuration, active control state, applied enforcement or forwarding, and end-to-end outcome independently.

Test normal, prohibited, degraded, failed, recovery, upgrade, and rollback conditions.

Preserve topology, identity, policy, configuration, packet, telemetry, log, timing, and service evidence.

Automate through typed data, deterministic gates, canaries, bounded authority, and reversible changes.

Failure patterns

Treating trust boundaries and residual exposure as a vendor feature instead of an end-to-end engineering system.

Relying on intended configuration or controller state without proving applied and operational behavior.

Ignoring reverse paths, external dependencies, identity, time, resource limits, or partial failure.

Changing several variables before preserving evidence and stating a falsifiable hypothesis.

Allowing automation or AI to act on stale, conflicting, unverified, or unauthorized inputs.

Declaring success after one positive probe without negative tests, scale checks, rollback, and recurrence monitoring.

Validation and evidence

Method	Expected evidence
Examine	Requirements, architecture, identities, models, policy, configuration, ownership, and dependencies for trust boundaries and residual exposure.
Observe	Control state, applied policy or forwarding, telemetry, logs, packets, sessions, resources, and endpoint behavior.
Test	Expected behavior, prohibited behavior, dependency loss, partial failure, scale, recovery, and rollback.
Measure	Availability, latency, loss, convergence, risk, resource use, change success, detection quality, and user or mission outcome.
Reconcile	Intended state against observed network, security, identity, application, data, and evidence records.

Chapter 01 laboratory and review

Practical laboratory

Draw site-to-site and remote-access trust models, including protected and exposed fields, identities, and dependencies.

Laboratory evidence package

- Architecture or topology showing boundaries, identities, dependencies, and authoritative inputs.
- Versioned configuration or code with explanatory comments and reproducible setup instructions.
- Nominal-path validation plus at least two injected failure or boundary conditions.
- Structured logs, traces, metrics, packet captures, test output, or other appropriate evidence.
- A concise decision record covering findings, limitations, residual risks, and next actions.

Frontier extension

Extend to quantum-safe key establishment, confidential gateways, verifiable policy, and per-application tunnels.

Review gate

Advance only when another engineer can reproduce the result, explain the architecture, identify the failure boundary, and distinguish measured evidence from assumptions or projections.

CHAPTER 02

IKEv2 negotiation and state

Trace exchanges that authenticate peers and create child security associations.

Chapter operating position
Trace exchanges that authenticate peers and create child security associations.

2.1 IKE_SA_INIT

IKE_SA_INIT must be engineered as an observable mechanism rather than a product label or checklist item. Analyze proposals, key-establishment material, nonces, cookies, NAT detection, fragmentation constraints, and state. The design should name its authoritative inputs, state transitions, dependencies, limits, security boundaries, operational owner, telemetry, failure behavior, and acceptance criteria.

This guide traces the subject through the peer reachability, IKE exchange, authentication, child SA, selector, route, NAT, encapsulation, endpoint, and application chain. A configured feature, established session, successful control-plane exchange, or green dashboard is not sufficient proof. Intended state must be reconciled with applied state, real traffic or transactions, negative tests, degraded behavior, recovery, and the resulting service or security outcome.

A production implementation starts with a minimal known-good baseline and explicit invariants. Introduce one dependency or capability at a time, preserve before-state evidence, test positive and prohibited behavior, inject realistic failures, measure convergence or recovery, and retain a tested rollback. Automation must stop safely when authority, freshness, identity, capacity, or observed outcomes are ambiguous.

Troubleshooting locates the first divergence from the expected state machine or end-to-end transaction. Inspect both directions and all administrative boundaries; account for caching, eventual consistency, stale identity, time, recursive dependencies, asymmetric paths, encapsulation, hardware offload, partial failure, and missing telemetry. Closure requires a causal explanation, reproducible evidence, validated restoration, residual limitations, and prevention or monitoring actions.

Engineering practices

- Define authority, ownership, scope, dependencies, limits, and acceptance criteria for ike_sa_init.
- Model the expected state, trust boundaries, and transaction before implementation or troubleshooting.
- Validate intended configuration, active control state, applied enforcement or forwarding, and end-to-end outcome independently.
- Test normal, prohibited, degraded, failed, recovery, upgrade, and rollback conditions.
- Preserve topology, identity, policy, configuration, packet, telemetry, log, timing, and service evidence.
- Automate through typed data, deterministic gates, canaries, bounded authority, and reversible changes.

Failure patterns

- Treating ike_sa_init as a vendor feature instead of an end-to-end engineering system.
- Relying on intended configuration or controller state without proving applied and operational behavior.
- Ignoring reverse paths, external dependencies, identity, time, resource limits, or partial failure.
- Changing several variables before preserving evidence and stating a falsifiable hypothesis.
- Allowing automation or AI to act on stale, conflicting, unverified, or unauthorized inputs.
- Declaring success after one positive probe without negative tests, scale checks, rollback, and recurrence monitoring.

Validation and evidence

Method	Expected evidence
Examine	Requirements, architecture, identities, models, policy, configuration, ownership, and dependencies for ike_sa_init.
Observe	Control state, applied policy or forwarding, telemetry, logs, packets, sessions, resources, and endpoint behavior.
Test	Expected behavior, prohibited behavior, dependency loss, partial failure, scale, recovery, and rollback.
Measure	Availability, latency, loss, convergence, risk, resource use, change success, detection quality, and user or mission outcome.
Reconcile	Intended state against observed network, security, identity, application, data, and evidence records.

2.2 IKE_AUTH

IKE_AUTH must be engineered as an observable mechanism rather than a product label or checklist item. Authenticate peers, validate identities and certificates, authorize traffic, exchange configuration, and create a child SA. The design should name its authoritative inputs, state transitions, dependencies, limits, security boundaries, operational owner, telemetry, failure behavior, and acceptance criteria.

This guide traces the subject through the peer reachability, IKE exchange, authentication, child SA, selector, route, NAT, encapsulation, endpoint, and application chain. A configured feature, established session, successful control-plane exchange, or green dashboard is not sufficient proof. Intended state must be reconciled with applied state, real traffic or transactions, negative tests, degraded behavior, recovery, and the resulting service or security outcome.

A production implementation starts with a minimal known-good baseline and explicit invariants. Introduce one dependency or capability at a time, preserve before-state evidence, test positive and prohibited behavior, inject realistic failures, measure convergence or recovery, and retain a tested rollback. Automation must stop safely when authority, freshness, identity, capacity, or observed outcomes are ambiguous.

Troubleshooting locates the first divergence from the expected state machine or end-to-end transaction. Inspect both directions and all administrative boundaries; account for caching, eventual consistency, stale identity, time, recursive dependencies, asymmetric paths, encapsulation, hardware offload, partial failure, and missing telemetry. Closure requires a causal explanation, reproducible evidence, validated restoration, residual limitations, and prevention or monitoring actions.

Engineering practices

- Define authority, ownership, scope, dependencies, limits, and acceptance criteria for ike_auth.
- Model the expected state, trust boundaries, and transaction before implementation or troubleshooting.
- Validate intended configuration, active control state, applied enforcement or forwarding, and end-to-end outcome independently.
- Test normal, prohibited, degraded, failed, recovery, upgrade, and rollback conditions.
- Preserve topology, identity, policy, configuration, packet, telemetry, log, timing, and service evidence.
- Automate through typed data, deterministic gates, canaries, bounded authority, and reversible changes.

Implementation sketch

```
ikev2_profile:
  authentication: certificate
  proposals:
    encryption: aes-gcm-256
    prf: hmac-sha2-384
    key_exchange: ecp384
  transition:
    multiple_key_exchange: evaluate
    ppk: controlled-pilot
    fragmentation: enabled
  child_sa:
    local: 10.10.0.0/16
    remote: 10.20.0.0/16
  validation:
    - ike_auth_identity
    - selectors_and_routes
    - nat_and_mtu
    - rekey_and_failover
```

Failure patterns

Treating ike_auth as a vendor feature instead of an end-to-end engineering system.

Relying on intended configuration or controller state without proving applied and operational behavior.

Ignoring reverse paths, external dependencies, identity, time, resource limits, or partial failure.

Changing several variables before preserving evidence and stating a falsifiable hypothesis.

Allowing automation or AI to act on stale, conflicting, unverified, or unauthorized inputs.

Declaring success after one positive probe without negative tests, scale checks, rollback, and recurrence monitoring.

Validation and evidence

Method	Expected evidence
Examine	Requirements, architecture, identities, models, policy, configuration, ownership, and dependencies for ike_auth.
Observe	Control state, applied policy or forwarding, telemetry, logs, packets, sessions, resources, and endpoint behavior.
Test	Expected behavior, prohibited behavior, dependency loss, partial failure, scale, recovery, and rollback.
Measure	Availability, latency, loss, convergence, risk, resource use, change success, detection quality, and user or mission outcome.
Reconcile	Intended state against observed network, security, identity, application, data, and evidence records.

2.3 CREATE_CHILD_SA and INFORMATIONAL

CREATE_CHILD_SA and INFORMATIONAL must be engineered as an observable mechanism rather than a product label or checklist item. Rekey SAs, create selectors, delete state, report errors, and perform liveness checks. The design should name its authoritative inputs, state transitions, dependencies, limits, security boundaries, operational owner, telemetry, failure behavior, and acceptance criteria.

This guide traces the subject through the peer reachability, IKE exchange, authentication, child SA, selector, route, NAT, encapsulation, endpoint, and application chain. A configured feature, established session, successful control-plane exchange, or green dashboard is not sufficient proof. Intended state must be reconciled with applied state, real traffic or transactions, negative tests, degraded behavior, recovery, and the resulting service or security outcome.

A production implementation starts with a minimal known-good baseline and explicit invariants. Introduce one dependency or capability at a time, preserve before-state evidence, test positive and prohibited behavior, inject realistic failures, measure convergence or recovery, and retain a tested rollback. Automation must stop safely when authority, freshness, identity, capacity, or observed outcomes are ambiguous.

Troubleshooting locates the first divergence from the expected state machine or end-to-end transaction. Inspect both directions and all administrative boundaries; account for caching, eventual consistency, stale identity, time, recursive dependencies,

asymmetric paths, encapsulation, hardware offload, partial failure, and missing telemetry. Closure requires a causal explanation, reproducible evidence, validated restoration, residual limitations, and prevention or monitoring actions.

Engineering practices

Define authority, ownership, scope, dependencies, limits, and acceptance criteria for create_child_sa and informational.

Model the expected state, trust boundaries, and transaction before implementation or troubleshooting.

Validate intended configuration, active control state, applied enforcement or forwarding, and end-to-end outcome independently.

Test normal, prohibited, degraded, failed, recovery, upgrade, and rollback conditions.

Preserve topology, identity, policy, configuration, packet, telemetry, log, timing, and service evidence.

Automate through typed data, deterministic gates, canaries, bounded authority, and reversible changes.

Failure patterns

Treating create_child_sa and informational as a vendor feature instead of an end-to-end engineering system.

Relying on intended configuration or controller state without proving applied and operational behavior.

Ignoring reverse paths, external dependencies, identity, time, resource limits, or partial failure.

Changing several variables before preserving evidence and stating a falsifiable hypothesis.

Allowing automation or AI to act on stale, conflicting, unverified, or unauthorized inputs.

Declaring success after one positive probe without negative tests, scale checks, rollback, and recurrence monitoring.

Validation and evidence

Method	Expected evidence
Examine	Requirements, architecture, identities, models, policy, configuration, ownership, and dependencies for create_child_sa and informational.
Observe	Control state, applied policy or forwarding, telemetry, logs, packets, sessions, resources, and endpoint behavior.
Test	Expected behavior, prohibited behavior, dependency loss, partial failure, scale, recovery, and rollback.
Measure	Availability, latency, loss, convergence, risk, resource use, change success, detection quality, and user or mission outcome.
Reconcile	Intended state against observed network, security, identity, application, data, and evidence records.

2.4 State, timers, and rekey collisions

State, timers, and rekey collisions must be engineered as an observable mechanism rather than a product label or checklist item. Manage lifetimes, overlapping SAs, retransmission, dead-peer detection, simultaneous rekey, and recovery. The design should name its authoritative inputs, state transitions, dependencies, limits, security boundaries, operational owner, telemetry, failure behavior, and acceptance criteria.

This guide traces the subject through the peer reachability, IKE exchange, authentication, child SA, selector, route, NAT, encapsulation, endpoint, and application chain. A configured feature, established session, successful control-plane exchange, or green dashboard is not sufficient proof. Intended state must be reconciled with applied state, real traffic or transactions, negative tests, degraded behavior, recovery, and the resulting service or security outcome.

A production implementation starts with a minimal known-good baseline and explicit invariants. Introduce one dependency or capability at a time, preserve before-state evidence, test positive and prohibited behavior, inject realistic failures, measure convergence or recovery, and retain a tested rollback. Automation must stop safely when authority, freshness, identity, capacity, or observed outcomes are ambiguous.

Troubleshooting locates the first divergence from the expected state machine or end-to-end transaction. Inspect both directions and all administrative boundaries; account for caching, eventual consistency, stale identity, time, recursive dependencies, asymmetric paths, encapsulation, hardware offload, partial failure, and missing telemetry. Closure requires a causal explanation, reproducible evidence, validated restoration, residual limitations, and prevention or monitoring actions.

Engineering practices

Define authority, ownership, scope, dependencies, limits, and acceptance criteria for state, timers, and rekey collisions.

Model the expected state, trust boundaries, and transaction before implementation or troubleshooting.

Validate intended configuration, active control state, applied enforcement or forwarding, and end-to-end outcome independently.

Test normal, prohibited, degraded, failed, recovery, upgrade, and rollback conditions.

Preserve topology, identity, policy, configuration, packet, telemetry, log, timing, and service evidence.

Automate through typed data, deterministic gates, canaries, bounded authority, and reversible changes.

Failure patterns

Treating state, timers, and rekey collisions as a vendor feature instead of an end-to-end engineering system.

Relying on intended configuration or controller state without proving applied and operational behavior.

Ignoring reverse paths, external dependencies, identity, time, resource limits, or partial failure.

Changing several variables before preserving evidence and stating a falsifiable hypothesis.

Allowing automation or AI to act on stale, conflicting, unverified, or unauthorized inputs.

Declaring success after one positive probe without negative tests, scale checks, rollback, and recurrence monitoring.

Validation and evidence

Method	Expected evidence
Examine	Requirements, architecture, identities, models, policy, configuration, ownership, and dependencies for state, timers, and rekey collisions.
Observe	Control state, applied policy or forwarding, telemetry, logs, packets, sessions, resources, and endpoint behavior.
Test	Expected behavior, prohibited behavior, dependency loss, partial failure, scale, recovery, and rollback.
Measure	Availability, latency, loss, convergence, risk, resource use, change success, detection quality, and user or mission outcome.
Reconcile	Intended state against observed network, security, identity, application, data, and evidence records.

Chapter 02 laboratory and review

Practical laboratory
Capture establishment, rekey, liveness, and deletion; annotate payloads and correlate with gateway state.

Laboratory evidence package

- Architecture or topology showing boundaries, identities, dependencies, and authoritative inputs.
- Versioned configuration or code with explanatory comments and reproducible setup instructions.
- Nominal-path validation plus at least two injected failure or boundary conditions.
- Structured logs, traces, metrics, packet captures, test output, or other appropriate evidence.
- A concise decision record covering findings, limitations, residual risks, and next actions.

Frontier extension

Explore multiple key exchanges, intermediate exchanges, PQ preshared keys, and larger hybrid messages.

Review gate
Advance only when another engineer can reproduce the result, explain the architecture, identify the failure boundary, and distinguish measured evidence from assumptions or projections.

CHAPTER 03

Cryptography, certificates, and authentication

Engineer algorithm and identity lifecycle instead of relying on defaults.

Chapter operating position

Engineer algorithm and identity lifecycle instead of relying on defaults.

3.1 Algorithm selection and agility

Algorithm selection and agility must be engineered as an observable mechanism rather than a product label or checklist item. Choose AEAD, PRFs, hashes, groups, signatures, key lengths, lifetimes, and transition policy. The design should name its authoritative inputs, state transitions, dependencies, limits, security boundaries, operational owner, telemetry, failure behavior, and acceptance criteria.

This guide traces the subject through the peer reachability, IKE exchange, authentication, child SA, selector, route, NAT, encapsulation, endpoint, and application chain. A configured feature, established session, successful control-plane exchange, or green dashboard is not sufficient proof. Intended state must be reconciled with applied state, real traffic or transactions, negative tests, degraded behavior, recovery, and the resulting service or security outcome.

A production implementation starts with a minimal known-good baseline and explicit invariants. Introduce one dependency or capability at a time, preserve before-state evidence, test positive and prohibited behavior, inject realistic failures, measure convergence or recovery, and retain a tested rollback. Automation must stop safely when authority, freshness, identity, capacity, or observed outcomes are ambiguous.

Troubleshooting locates the first divergence from the expected state machine or end-to-end transaction. Inspect both directions and all administrative boundaries; account for caching, eventual consistency, stale identity, time, recursive dependencies, asymmetric paths, encapsulation, hardware offload, partial failure, and missing telemetry. Closure requires a causal explanation, reproducible evidence, validated restoration, residual limitations, and prevention or monitoring actions.

Engineering practices

- Define authority, ownership, scope, dependencies, limits, and acceptance criteria for algorithm selection and agility.
- Model the expected state, trust boundaries, and transaction before implementation or troubleshooting.
- Validate intended configuration, active control state, applied enforcement or forwarding, and end-to-end outcome independently.
- Test normal, prohibited, degraded, failed, recovery, upgrade, and rollback conditions.
- Preserve topology, identity, policy, configuration, packet, telemetry, log, timing, and service evidence.
- Automate through typed data, deterministic gates, canaries, bounded authority, and reversible changes.

Failure patterns

- Treating algorithm selection and agility as a vendor feature instead of an end-to-end engineering system.
- Relying on intended configuration or controller state without proving applied and operational behavior.
- Ignoring reverse paths, external dependencies, identity, time, resource limits, or partial failure.
- Changing several variables before preserving evidence and stating a falsifiable hypothesis.
- Allowing automation or AI to act on stale, conflicting, unverified, or unauthorized inputs.
- Declaring success after one positive probe without negative tests, scale checks, rollback, and recurrence monitoring.

Validation and evidence

Method	Expected evidence
Examine	Requirements, architecture, identities, models, policy, configuration, ownership, and dependencies for algorithm selection and agility.
Observe	Control state, applied policy or forwarding, telemetry, logs, packets, sessions, resources, and endpoint behavior.
Test	Expected behavior, prohibited behavior, dependency loss, partial failure, scale, recovery, and rollback.
Measure	Availability, latency, loss, convergence, risk, resource use, change success, detection quality, and user or mission outcome.
Reconcile	Intended state against observed network, security, identity, application, data, and evidence records.

3.2 Certificate-based authentication

Certificate-based authentication must be engineered as an observable mechanism rather than a product label or checklist item. Validate identity, SAN, chain, EKU, revocation, time, renewal, intermediates, and hardware keys. The design should name its authoritative inputs, state transitions, dependencies, limits, security boundaries, operational owner, telemetry, failure behavior, and acceptance criteria.

This guide traces the subject through the peer reachability, IKE exchange, authentication, child SA, selector, route, NAT, encapsulation, endpoint, and application chain. A configured feature, established session, successful control-plane exchange, or green dashboard is not sufficient proof. Intended state must be reconciled with applied state, real traffic or transactions, negative tests, degraded behavior, recovery, and the resulting service or security outcome.

A production implementation starts with a minimal known-good baseline and explicit invariants. Introduce one dependency or capability at a time, preserve before-state evidence, test positive and prohibited behavior, inject realistic failures, measure convergence or recovery, and retain a tested rollback. Automation must stop safely when authority, freshness, identity, capacity, or observed outcomes are ambiguous.

Troubleshooting locates the first divergence from the expected state machine or end-to-end transaction. Inspect both directions and all administrative boundaries; account for caching, eventual consistency, stale identity, time, recursive dependencies, asymmetric paths, encapsulation, hardware offload, partial failure, and missing telemetry. Closure requires a causal explanation, reproducible evidence, validated restoration, residual limitations, and prevention or monitoring actions.

Engineering practices

- Define authority, ownership, scope, dependencies, limits, and acceptance criteria for certificate-based authentication.
- Model the expected state, trust boundaries, and transaction before implementation or troubleshooting.
- Validate intended configuration, active control state, applied enforcement or forwarding, and end-to-end outcome independently.
- Test normal, prohibited, degraded, failed, recovery, upgrade, and rollback conditions.
- Preserve topology, identity, policy, configuration, packet, telemetry, log, timing, and service evidence.
- Automate through typed data, deterministic gates, canaries, bounded authority, and reversible changes.

Implementation sketch

```
ikev2_profile:
  authentication: certificate
  proposals:
    encryption: aes-gcm-256
    prf: hmac-sha2-384
    key_exchange: ecp384
  transition:
    multiple_key_exchange: evaluate
    ppk: controlled-pilot
    fragmentation: enabled
  child_sa:
    local: 10.10.0.0/16
    remote: 10.20.0.0/16
  validation:
    - ike_auth_identity
    - selectors_and_routes
    - nat_and_mtu
    - rekey_and_failover
```

Failure patterns

Treating certificate-based authentication as a vendor feature instead of an end-to-end engineering system.

Relying on intended configuration or controller state without proving applied and operational behavior.

Ignoring reverse paths, external dependencies, identity, time, resource limits, or partial failure.

Changing several variables before preserving evidence and stating a falsifiable hypothesis.

Allowing automation or AI to act on stale, conflicting, unverified, or unauthorized inputs.

Declaring success after one positive probe without negative tests, scale checks, rollback, and recurrence monitoring.

Validation and evidence

Method	Expected evidence
Examine	Requirements, architecture, identities, models, policy, configuration, ownership, and dependencies for certificate-based authentication.
Observe	Control state, applied policy or forwarding, telemetry, logs, packets, sessions, resources, and endpoint behavior.
Test	Expected behavior, prohibited behavior, dependency loss, partial failure, scale, recovery, and rollback.
Measure	Availability, latency, loss, convergence, risk, resource use, change success, detection quality, and user or mission outcome.
Reconcile	Intended state against observed network, security, identity, application, data, and evidence records.

3.3 Preshared keys and external authentication

Preshared keys and external authentication must be engineered as an observable mechanism rather than a product label or checklist item. Assess entropy, distribution, uniqueness, rotation, scaling, EAP, federation, MFA, and recovery. The design should name its authoritative inputs, state transitions, dependencies, limits, security boundaries, operational owner, telemetry, failure behavior, and acceptance criteria.

This guide traces the subject through the peer reachability, IKE exchange, authentication, child SA, selector, route, NAT, encapsulation, endpoint, and application chain. A configured feature, established session, successful control-plane exchange, or green dashboard is not sufficient proof. Intended state must be reconciled with applied state, real traffic or transactions, negative tests, degraded behavior, recovery, and the resulting service or security outcome.

A production implementation starts with a minimal known-good baseline and explicit invariants. Introduce one dependency or capability at a time, preserve before-state evidence, test positive and prohibited behavior, inject realistic failures, measure convergence or recovery, and retain a tested rollback. Automation must stop safely when authority, freshness, identity, capacity, or observed outcomes are ambiguous.

Troubleshooting locates the first divergence from the expected state machine or end-to-end transaction. Inspect both directions and all administrative boundaries; account for caching, eventual consistency, stale identity, time, recursive dependencies,

asymmetric paths, encapsulation, hardware offload, partial failure, and missing telemetry. Closure requires a causal explanation, reproducible evidence, validated restoration, residual limitations, and prevention or monitoring actions.

Engineering practices

Define authority, ownership, scope, dependencies, limits, and acceptance criteria for preshared keys and external authentication.

Model the expected state, trust boundaries, and transaction before implementation or troubleshooting.

Validate intended configuration, active control state, applied enforcement or forwarding, and end-to-end outcome independently.

Test normal, prohibited, degraded, failed, recovery, upgrade, and rollback conditions.

Preserve topology, identity, policy, configuration, packet, telemetry, log, timing, and service evidence.

Automate through typed data, deterministic gates, canaries, bounded authority, and reversible changes.

Failure patterns

Treating preshared keys and external authentication as a vendor feature instead of an end-to-end engineering system.

Relying on intended configuration or controller state without proving applied and operational behavior.

Ignoring reverse paths, external dependencies, identity, time, resource limits, or partial failure.

Changing several variables before preserving evidence and stating a falsifiable hypothesis.

Allowing automation or AI to act on stale, conflicting, unverified, or unauthorized inputs.

Declaring success after one positive probe without negative tests, scale checks, rollback, and recurrence monitoring.

Validation and evidence

Method	Expected evidence
Examine	Requirements, architecture, identities, models, policy, configuration, ownership, and dependencies for preshared keys and external authentication.
Observe	Control state, applied policy or forwarding, telemetry, logs, packets, sessions, resources, and endpoint behavior.
Test	Expected behavior, prohibited behavior, dependency loss, partial failure, scale, recovery, and rollback.
Measure	Availability, latency, loss, convergence, risk, resource use, change success, detection quality, and user or mission outcome.
Reconcile	Intended state against observed network, security, identity, application, data, and evidence records.

3.4 Post-quantum transition

Post-quantum transition must be engineered as an observable mechanism rather than a product label or checklist item. Inventory dependencies, evaluate hybrid or PPK options, message size, fragmentation, interoperability, and long-lived data risk. The design should name its authoritative inputs, state transitions, dependencies, limits, security boundaries, operational owner, telemetry, failure behavior, and acceptance criteria.

This guide traces the subject through the peer reachability, IKE exchange, authentication, child SA, selector, route, NAT, encapsulation, endpoint, and application chain. A configured feature, established session, successful control-plane exchange, or green dashboard is not sufficient proof. Intended state must be reconciled with applied state, real traffic or transactions, negative tests, degraded behavior, recovery, and the resulting service or security outcome.

A production implementation starts with a minimal known-good baseline and explicit invariants. Introduce one dependency or capability at a time, preserve before-state evidence, test positive and prohibited behavior, inject realistic failures, measure convergence or recovery, and retain a tested rollback. Automation must stop safely when authority, freshness, identity, capacity, or observed outcomes are ambiguous.

Troubleshooting locates the first divergence from the expected state machine or end-to-end transaction. Inspect both directions and all administrative boundaries; account for caching, eventual consistency, stale identity, time, recursive dependencies, asymmetric paths, encapsulation, hardware offload, partial failure, and missing telemetry. Closure requires a causal explanation, reproducible evidence, validated restoration, residual limitations, and prevention or monitoring actions.

Engineering practices

Define authority, ownership, scope, dependencies, limits, and acceptance criteria for post-quantum transition.

Model the expected state, trust boundaries, and transaction before implementation or troubleshooting.

Validate intended configuration, active control state, applied enforcement or forwarding, and end-to-end outcome independently.

Test normal, prohibited, degraded, failed, recovery, upgrade, and rollback conditions.

Preserve topology, identity, policy, configuration, packet, telemetry, log, timing, and service evidence.

Automate through typed data, deterministic gates, canaries, bounded authority, and reversible changes.

Failure patterns

Treating post-quantum transition as a vendor feature instead of an end-to-end engineering system.

Relying on intended configuration or controller state without proving applied and operational behavior.

Ignoring reverse paths, external dependencies, identity, time, resource limits, or partial failure.

Changing several variables before preserving evidence and stating a falsifiable hypothesis.

Allowing automation or AI to act on stale, conflicting, unverified, or unauthorized inputs.

Declaring success after one positive probe without negative tests, scale checks, rollback, and recurrence monitoring.

Validation and evidence

Method	Expected evidence
Examine	Requirements, architecture, identities, models, policy, configuration, ownership, and dependencies for post-quantum transition.
Observe	Control state, applied policy or forwarding, telemetry, logs, packets, sessions, resources, and endpoint behavior.
Test	Expected behavior, prohibited behavior, dependency loss, partial failure, scale, recovery, and rollback.
Measure	Availability, latency, loss, convergence, risk, resource use, change success, detection quality, and user or mission outcome.
Reconcile	Intended state against observed network, security, identity, application, data, and evidence records.

Chapter 03 laboratory and review

Practical laboratory
Build certificate and PSK profiles, introduce trust, identity, expiry, revocation, and time failures, and document exact negotiation evidence.

Laboratory evidence package

- Architecture or topology showing boundaries, identities, dependencies, and authoritative inputs.
- Versioned configuration or code with explanatory comments and reproducible setup instructions.
- Nominal-path validation plus at least two injected failure or boundary conditions.
- Structured logs, traces, metrics, packet captures, test output, or other appropriate evidence.
- A concise decision record covering findings, limitations, residual risks, and next actions.

Frontier extension

Evaluate ML-KEM-based or hybrid IKE designs as implementations mature, distinguishing standards from experiments.

Review gate
Advance only when another engineer can reproduce the result, explain the architecture, identify the failure boundary, and distinguish measured evidence from assumptions or projections.

CHAPTER 04

Site-to-site VPN engineering

Integrate encrypted transport with routing, NAT, HA, and policy.

Chapter operating position

Integrate encrypted transport with routing, NAT, HA, and policy.

4.1 Policy-based and route-based models

Policy-based and route-based models must be engineered as an observable mechanism rather than a product label or checklist item. Compare selectors, tunnel interfaces, routing, scale, multipath, dynamic routing, troubleshooting, and migration. The design should name its authoritative inputs, state transitions, dependencies, limits, security boundaries, operational owner, telemetry, failure behavior, and acceptance criteria.

This guide traces the subject through the peer reachability, IKE exchange, authentication, child SA, selector, route, NAT, encapsulation, endpoint, and application chain. A configured feature, established session, successful control-plane exchange, or green dashboard is not sufficient proof. Intended state must be reconciled with applied state, real traffic or transactions, negative tests, degraded behavior, recovery, and the resulting service or security outcome.

A production implementation starts with a minimal known-good baseline and explicit invariants. Introduce one dependency or capability at a time, preserve before-state evidence, test positive and prohibited behavior, inject realistic failures, measure convergence or recovery, and retain a tested rollback. Automation must stop safely when authority, freshness, identity, capacity, or observed outcomes are ambiguous.

Troubleshooting locates the first divergence from the expected state machine or end-to-end transaction. Inspect both directions and all administrative boundaries; account for caching, eventual consistency, stale identity, time, recursive dependencies, asymmetric paths, encapsulation, hardware offload, partial failure, and missing telemetry. Closure requires a causal explanation, reproducible evidence, validated restoration, residual limitations, and prevention or monitoring actions.

Engineering practices

- Define authority, ownership, scope, dependencies, limits, and acceptance criteria for policy-based and route-based models.
- Model the expected state, trust boundaries, and transaction before implementation or troubleshooting.
- Validate intended configuration, active control state, applied enforcement or forwarding, and end-to-end outcome independently.
- Test normal, prohibited, degraded, failed, recovery, upgrade, and rollback conditions.
- Preserve topology, identity, policy, configuration, packet, telemetry, log, timing, and service evidence.
- Automate through typed data, deterministic gates, canaries, bounded authority, and reversible changes.

Failure patterns

- Treating policy-based and route-based models as a vendor feature instead of an end-to-end engineering system.
- Relying on intended configuration or controller state without proving applied and operational behavior.
- Ignoring reverse paths, external dependencies, identity, time, resource limits, or partial failure.
- Changing several variables before preserving evidence and stating a falsifiable hypothesis.
- Allowing automation or AI to act on stale, conflicting, unverified, or unauthorized inputs.
- Declaring success after one positive probe without negative tests, scale checks, rollback, and recurrence monitoring.

Validation and evidence

Method	Expected evidence
Examine	Requirements, architecture, identities, models, policy, configuration, ownership, and dependencies for policy-based and route-based models.
Observe	Control state, applied policy or forwarding, telemetry, logs, packets, sessions, resources, and endpoint behavior.
Test	Expected behavior, prohibited behavior, dependency loss, partial failure, scale, recovery, and rollback.
Measure	Availability, latency, loss, convergence, risk, resource use, change success, detection quality, and user or mission outcome.
Reconcile	Intended state against observed network, security, identity, application, data, and evidence records.

4.2 Traffic selectors and proxy identities

Traffic selectors and proxy identities must be engineered as an observable mechanism rather than a product label or checklist item. Match local and remote networks, narrowing, overlap, multiple child SAs, symmetry, and compatibility. The design should name its authoritative inputs, state transitions, dependencies, limits, security boundaries, operational owner, telemetry, failure behavior, and acceptance criteria.

This guide traces the subject through the peer reachability, IKE exchange, authentication, child SA, selector, route, NAT, encapsulation, endpoint, and application chain. A configured feature, established session, successful control-plane exchange, or green dashboard is not sufficient proof. Intended state must be reconciled with applied state, real traffic or transactions, negative tests, degraded behavior, recovery, and the resulting service or security outcome.

A production implementation starts with a minimal known-good baseline and explicit invariants. Introduce one dependency or capability at a time, preserve before-state evidence, test positive and prohibited behavior, inject realistic failures, measure convergence or recovery, and retain a tested rollback. Automation must stop safely when authority, freshness, identity, capacity, or observed outcomes are ambiguous.

Troubleshooting locates the first divergence from the expected state machine or end-to-end transaction. Inspect both directions and all administrative boundaries; account for caching, eventual consistency, stale identity, time, recursive dependencies, asymmetric paths, encapsulation, hardware offload, partial failure, and missing telemetry. Closure requires a causal explanation, reproducible evidence, validated restoration, residual limitations, and prevention or monitoring actions.

Engineering practices

Define authority, ownership, scope, dependencies, limits, and acceptance criteria for traffic selectors and proxy identities.

Model the expected state, trust boundaries, and transaction before implementation or troubleshooting.

Validate intended configuration, active control state, applied enforcement or forwarding, and end-to-end outcome independently.

Test normal, prohibited, degraded, failed, recovery, upgrade, and rollback conditions.

Preserve topology, identity, policy, configuration, packet, telemetry, log, timing, and service evidence.

Automate through typed data, deterministic gates, canaries, bounded authority, and reversible changes.

Implementation sketch

```
ikev2_profile:
  authentication: certificate
  proposals:
    encryption: aes-gcm-256
    prf: hmac-sha2-384
    key_exchange: ecp384
  transition:
    multiple_key_exchange: evaluate
    ppk: controlled-pilot
    fragmentation: enabled
  child_sa:
    local: 10.10.0.0/16
    remote: 10.20.0.0/16
  validation:
    - ike_auth_identity
    - selectors_and_routes
    - nat_and_mtu
    - rekey_and_failover
```

Failure patterns

Treating traffic selectors and proxy identities as a vendor feature instead of an end-to-end engineering system.

Relying on intended configuration or controller state without proving applied and operational behavior.

Ignoring reverse paths, external dependencies, identity, time, resource limits, or partial failure.

Changing several variables before preserving evidence and stating a falsifiable hypothesis.

Allowing automation or AI to act on stale, conflicting, unverified, or unauthorized inputs.

Declaring success after one positive probe without negative tests, scale checks, rollback, and recurrence monitoring.

Validation and evidence

Method	Expected evidence
Examine	Requirements, architecture, identities, models, policy, configuration, ownership, and dependencies for traffic selectors and proxy identities.
Observe	Control state, applied policy or forwarding, telemetry, logs, packets, sessions, resources, and endpoint behavior.
Test	Expected behavior, prohibited behavior, dependency loss, partial failure, scale, recovery, and rollback.
Measure	Availability, latency, loss, convergence, risk, resource use, change success, detection quality, and user or mission outcome.
Reconcile	Intended state against observed network, security, identity, application, data, and evidence records.

4.3 NAT traversal and overlap

NAT traversal and overlap must be engineered as an observable mechanism rather than a product label or checklist item. Understand UDP encapsulation, detection, keepalives, translation, no-NAT, overlap, and logging. The design should name its authoritative inputs, state transitions, dependencies, limits, security boundaries, operational owner, telemetry, failure behavior, and acceptance criteria.

This guide traces the subject through the peer reachability, IKE exchange, authentication, child SA, selector, route, NAT, encapsulation, endpoint, and application chain. A configured feature, established session, successful control-plane exchange, or green dashboard is not sufficient proof. Intended state must be reconciled with applied state, real traffic or transactions, negative tests, degraded behavior, recovery, and the resulting service or security outcome.

A production implementation starts with a minimal known-good baseline and explicit invariants. Introduce one dependency or capability at a time, preserve before-state evidence, test positive and prohibited behavior, inject realistic failures, measure convergence or recovery, and retain a tested rollback. Automation must stop safely when authority, freshness, identity, capacity, or observed outcomes are ambiguous.

Troubleshooting locates the first divergence from the expected state machine or end-to-end transaction. Inspect both directions and all administrative boundaries; account for caching, eventual consistency, stale identity, time, recursive dependencies,

asymmetric paths, encapsulation, hardware offload, partial failure, and missing telemetry. Closure requires a causal explanation, reproducible evidence, validated restoration, residual limitations, and prevention or monitoring actions.

Engineering practices

Define authority, ownership, scope, dependencies, limits, and acceptance criteria for nat traversal and overlap.

Model the expected state, trust boundaries, and transaction before implementation or troubleshooting.

Validate intended configuration, active control state, applied enforcement or forwarding, and end-to-end outcome independently.

Test normal, prohibited, degraded, failed, recovery, upgrade, and rollback conditions.

Preserve topology, identity, policy, configuration, packet, telemetry, log, timing, and service evidence.

Automate through typed data, deterministic gates, canaries, bounded authority, and reversible changes.

Failure patterns

Treating nat traversal and overlap as a vendor feature instead of an end-to-end engineering system.

Relying on intended configuration or controller state without proving applied and operational behavior.

Ignoring reverse paths, external dependencies, identity, time, resource limits, or partial failure.

Changing several variables before preserving evidence and stating a falsifiable hypothesis.

Allowing automation or AI to act on stale, conflicting, unverified, or unauthorized inputs.

Declaring success after one positive probe without negative tests, scale checks, rollback, and recurrence monitoring.

Validation and evidence

Method	Expected evidence
Examine	Requirements, architecture, identities, models, policy, configuration, ownership, and dependencies for nat traversal and overlap.
Observe	Control state, applied policy or forwarding, telemetry, logs, packets, sessions, resources, and endpoint behavior.
Test	Expected behavior, prohibited behavior, dependency loss, partial failure, scale, recovery, and rollback.
Measure	Availability, latency, loss, convergence, risk, resource use, change success, detection quality, and user or mission outcome.
Reconcile	Intended state against observed network, security, identity, application, data, and evidence records.

4.4 Routing and failure behavior

Routing and failure behavior must be engineered as an observable mechanism rather than a product label or checklist item. Use static or dynamic routes, monitoring, BFD, path selection, black-hole prevention, and recovery. The design should name its authoritative inputs, state transitions, dependencies, limits, security boundaries, operational owner, telemetry, failure behavior, and acceptance criteria.

This guide traces the subject through the peer reachability, IKE exchange, authentication, child SA, selector, route, NAT, encapsulation, endpoint, and application chain. A configured feature, established session, successful control-plane exchange, or green dashboard is not sufficient proof. Intended state must be reconciled with applied state, real traffic or transactions, negative tests, degraded behavior, recovery, and the resulting service or security outcome.

A production implementation starts with a minimal known-good baseline and explicit invariants. Introduce one dependency or capability at a time, preserve before-state evidence, test positive and prohibited behavior, inject realistic failures, measure convergence or recovery, and retain a tested rollback. Automation must stop safely when authority, freshness, identity, capacity, or observed outcomes are ambiguous.

Troubleshooting locates the first divergence from the expected state machine or end-to-end transaction. Inspect both directions and all administrative boundaries; account for caching, eventual consistency, stale identity, time, recursive dependencies, asymmetric paths, encapsulation, hardware offload, partial failure, and missing telemetry. Closure requires a causal explanation, reproducible evidence, validated restoration, residual limitations, and prevention or monitoring actions.

Engineering practices

Define authority, ownership, scope, dependencies, limits, and acceptance criteria for routing and failure behavior.

Model the expected state, trust boundaries, and transaction before implementation or troubleshooting.

Validate intended configuration, active control state, applied enforcement or forwarding, and end-to-end outcome independently.

Test normal, prohibited, degraded, failed, recovery, upgrade, and rollback conditions.

Preserve topology, identity, policy, configuration, packet, telemetry, log, timing, and service evidence.

Automate through typed data, deterministic gates, canaries, bounded authority, and reversible changes.

Failure patterns

Treating routing and failure behavior as a vendor feature instead of an end-to-end engineering system.

Relying on intended configuration or controller state without proving applied and operational behavior.

Ignoring reverse paths, external dependencies, identity, time, resource limits, or partial failure.

Changing several variables before preserving evidence and stating a falsifiable hypothesis.

Allowing automation or AI to act on stale, conflicting, unverified, or unauthorized inputs.

Declaring success after one positive probe without negative tests, scale checks, rollback, and recurrence monitoring.

Validation and evidence

Method	Expected evidence
Examine	Requirements, architecture, identities, models, policy, configuration, ownership, and dependencies for routing and failure behavior.
Observe	Control state, applied policy or forwarding, telemetry, logs, packets, sessions, resources, and endpoint behavior.
Test	Expected behavior, prohibited behavior, dependency loss, partial failure, scale, recovery, and rollback.
Measure	Availability, latency, loss, convergence, risk, resource use, change success, detection quality, and user or mission outcome.
Reconcile	Intended state against observed network, security, identity, application, data, and evidence records.

Chapter 04 laboratory and review

Practical laboratory
Build a route-based VPN with dynamic routing and a policy-based comparison; test selectors, NAT, path loss, overlap, and asymmetry.

Laboratory evidence package

- Architecture or topology showing boundaries, identities, dependencies, and authoritative inputs.
- Versioned configuration or code with explanatory comments and reproducible setup instructions.
- Nominal-path validation plus at least two injected failure or boundary conditions.
- Structured logs, traces, metrics, packet captures, test output, or other appropriate evidence.
- A concise decision record covering findings, limitations, residual risks, and next actions.

Frontier extension

Explore SD-WAN-integrated IPsec, multi-cloud fabrics, per-service tunnels, and intent-driven cryptographic routing.

Review gate
Advance only when another engineer can reproduce the result, explain the architecture, identify the failure boundary, and distinguish measured evidence from assumptions or projections.

CHAPTER 05

Remote access and TLS VPN

Design user and device access from authentication through application authorization.

Chapter operating position

Design user and device access from authentication through application authorization.

5.1 Client, portal, gateway, and broker architecture

Client, portal, gateway, and broker architecture must be engineered as an observable mechanism rather than a product label or checklist item. Separate discovery, authentication, posture, tunnel, policy, application access, and evidence. The design should name its authoritative inputs, state transitions, dependencies, limits, security boundaries, operational owner, telemetry, failure behavior, and acceptance criteria.

This guide traces the subject through the peer reachability, IKE exchange, authentication, child SA, selector, route, NAT, encapsulation, endpoint, and application chain. A configured feature, established session, successful control-plane exchange, or green dashboard is not sufficient proof. Intended state must be reconciled with applied state, real traffic or transactions, negative tests, degraded behavior, recovery, and the resulting service or security outcome.

A production implementation starts with a minimal known-good baseline and explicit invariants. Introduce one dependency or capability at a time, preserve before-state evidence, test positive and prohibited behavior, inject realistic failures, measure convergence or recovery, and retain a tested rollback. Automation must stop safely when authority, freshness, identity, capacity, or observed outcomes are ambiguous.

Troubleshooting locates the first divergence from the expected state machine or end-to-end transaction. Inspect both directions and all administrative boundaries; account for caching, eventual consistency, stale identity, time, recursive dependencies, asymmetric paths, encapsulation, hardware offload, partial failure, and missing telemetry. Closure requires a causal explanation, reproducible evidence, validated restoration, residual limitations, and prevention or monitoring actions.

Engineering practices

- Define authority, ownership, scope, dependencies, limits, and acceptance criteria for client, portal, gateway, and broker architecture.
- Model the expected state, trust boundaries, and transaction before implementation or troubleshooting.
- Validate intended configuration, active control state, applied enforcement or forwarding, and end-to-end outcome independently.
- Test normal, prohibited, degraded, failed, recovery, upgrade, and rollback conditions.
- Preserve topology, identity, policy, configuration, packet, telemetry, log, timing, and service evidence.
- Automate through typed data, deterministic gates, canaries, bounded authority, and reversible changes.

Failure patterns

- Treating client, portal, gateway, and broker architecture as a vendor feature instead of an end-to-end engineering system.
- Relying on intended configuration or controller state without proving applied and operational behavior.
- Ignoring reverse paths, external dependencies, identity, time, resource limits, or partial failure.
- Changing several variables before preserving evidence and stating a falsifiable hypothesis.
- Allowing automation or AI to act on stale, conflicting, unverified, or unauthorized inputs.
- Declaring success after one positive probe without negative tests, scale checks, rollback, and recurrence monitoring.

Validation and evidence

Method	Expected evidence
Examine	Requirements, architecture, identities, models, policy, configuration, ownership, and dependencies for client, portal, gateway, and broker architecture.
Observe	Control state, applied policy or forwarding, telemetry, logs, packets, sessions, resources, and endpoint behavior.
Test	Expected behavior, prohibited behavior, dependency loss, partial failure, scale, recovery, and rollback.
Measure	Availability, latency, loss, convergence, risk, resource use, change success, detection quality, and user or mission outcome.
Reconcile	Intended state against observed network, security, identity, application, data, and evidence records.

5.2 TLS tunnel and application proxy models

TLS tunnel and application proxy models must be engineered as an observable mechanism rather than a product label or checklist item. Compare full network tunnels, per-app access, reverse proxies, browser access, and private access. The design should name its authoritative inputs, state transitions, dependencies, limits, security boundaries, operational owner, telemetry, failure behavior, and acceptance criteria.

This guide traces the subject through the peer reachability, IKE exchange, authentication, child SA, selector, route, NAT, encapsulation, endpoint, and application chain. A configured feature, established session, successful control-plane exchange, or green dashboard is not sufficient proof. Intended state must be reconciled with applied state, real traffic or transactions, negative tests, degraded behavior, recovery, and the resulting service or security outcome.

A production implementation starts with a minimal known-good baseline and explicit invariants. Introduce one dependency or capability at a time, preserve before-state evidence, test positive and prohibited behavior, inject realistic failures, measure convergence or recovery, and retain a tested rollback. Automation must stop safely when authority, freshness, identity, capacity, or observed outcomes are ambiguous.

Troubleshooting locates the first divergence from the expected state machine or end-to-end transaction. Inspect both directions and all administrative boundaries; account for caching, eventual consistency, stale identity, time, recursive dependencies, asymmetric paths, encapsulation, hardware offload, partial failure, and missing telemetry. Closure requires a causal explanation, reproducible evidence, validated restoration, residual limitations, and prevention or monitoring actions.

Engineering practices

- Define authority, ownership, scope, dependencies, limits, and acceptance criteria for tls tunnel and application proxy models.
- Model the expected state, trust boundaries, and transaction before implementation or troubleshooting.
- Validate intended configuration, active control state, applied enforcement or forwarding, and end-to-end outcome independently.
- Test normal, prohibited, degraded, failed, recovery, upgrade, and rollback conditions.
- Preserve topology, identity, policy, configuration, packet, telemetry, log, timing, and service evidence.
- Automate through typed data, deterministic gates, canaries, bounded authority, and reversible changes.

Implementation sketch

```
ikev2_profile:
  authentication: certificate
  proposals:
    encryption: aes-gcm-256
    prf: hmac-sha2-384
    key_exchange: ecp384
  transition:
    multiple_key_exchange: evaluate
    ppk: controlled-pilot
    fragmentation: enabled
  child_sa:
    local: 10.10.0.0/16
    remote: 10.20.0.0/16
  validation:
    - ike_auth_identity
    - selectors_and_routes
    - nat_and_mtu
    - rekey_and_failover
```

Failure patterns

Treating tls tunnel and application proxy models as a vendor feature instead of an end-to-end engineering system.

Relying on intended configuration or controller state without proving applied and operational behavior.

Ignoring reverse paths, external dependencies, identity, time, resource limits, or partial failure.

Changing several variables before preserving evidence and stating a falsifiable hypothesis.

Allowing automation or AI to act on stale, conflicting, unverified, or unauthorized inputs.

Declaring success after one positive probe without negative tests, scale checks, rollback, and recurrence monitoring.

Validation and evidence

Method	Expected evidence
Examine	Requirements, architecture, identities, models, policy, configuration, ownership, and dependencies for tls tunnel and application proxy models.
Observe	Control state, applied policy or forwarding, telemetry, logs, packets, sessions, resources, and endpoint behavior.
Test	Expected behavior, prohibited behavior, dependency loss, partial failure, scale, recovery, and rollback.
Measure	Availability, latency, loss, convergence, risk, resource use, change success, detection quality, and user or mission outcome.
Reconcile	Intended state against observed network, security, identity, application, data, and evidence records.

5.3 Addressing, routes, DNS, and split tunneling

Addressing, routes, DNS, and split tunneling must be engineered as an observable mechanism rather than a product label or checklist item. Control pools, IPv4 and IPv6, route injection, exclusions, DNS, local LAN, SaaS, and leak prevention. The design should name its authoritative inputs, state transitions, dependencies, limits, security boundaries, operational owner, telemetry, failure behavior, and acceptance criteria.

This guide traces the subject through the peer reachability, IKE exchange, authentication, child SA, selector, route, NAT, encapsulation, endpoint, and application chain. A configured feature, established session, successful control-plane exchange, or green dashboard is not sufficient proof. Intended state must be reconciled with applied state, real traffic or transactions, negative tests, degraded behavior, recovery, and the resulting service or security outcome.

A production implementation starts with a minimal known-good baseline and explicit invariants. Introduce one dependency or capability at a time, preserve before-state evidence, test positive and prohibited behavior, inject realistic failures, measure convergence or recovery, and retain a tested rollback. Automation must stop safely when authority, freshness, identity, capacity, or observed outcomes are ambiguous.

Troubleshooting locates the first divergence from the expected state machine or end-to-end transaction. Inspect both directions and all administrative boundaries; account for caching, eventual consistency, stale identity, time, recursive dependencies,

asymmetric paths, encapsulation, hardware offload, partial failure, and missing telemetry. Closure requires a causal explanation, reproducible evidence, validated restoration, residual limitations, and prevention or monitoring actions.

Engineering practices

Define authority, ownership, scope, dependencies, limits, and acceptance criteria for addressing, routes, dns, and split tunneling.

Model the expected state, trust boundaries, and transaction before implementation or troubleshooting.

Validate intended configuration, active control state, applied enforcement or forwarding, and end-to-end outcome independently.

Test normal, prohibited, degraded, failed, recovery, upgrade, and rollback conditions.

Preserve topology, identity, policy, configuration, packet, telemetry, log, timing, and service evidence.

Automate through typed data, deterministic gates, canaries, bounded authority, and reversible changes.

Failure patterns

Treating addressing, routes, dns, and split tunneling as a vendor feature instead of an end-to-end engineering system.

Relying on intended configuration or controller state without proving applied and operational behavior.

Ignoring reverse paths, external dependencies, identity, time, resource limits, or partial failure.

Changing several variables before preserving evidence and stating a falsifiable hypothesis.

Allowing automation or AI to act on stale, conflicting, unverified, or unauthorized inputs.

Declaring success after one positive probe without negative tests, scale checks, rollback, and recurrence monitoring.

Validation and evidence

Method	Expected evidence
Examine	Requirements, architecture, identities, models, policy, configuration, ownership, and dependencies for addressing, routes, dns, and split tunneling.
Observe	Control state, applied policy or forwarding, telemetry, logs, packets, sessions, resources, and endpoint behavior.
Test	Expected behavior, prohibited behavior, dependency loss, partial failure, scale, recovery, and rollback.
Measure	Availability, latency, loss, convergence, risk, resource use, change success, detection quality, and user or mission outcome.
Reconcile	Intended state against observed network, security, identity, application, data, and evidence records.

5.4 Identity, MFA, posture, and device trust

Identity, MFA, posture, and device trust must be engineered as an observable mechanism rather than a product label or checklist item. Integrate user, device, certificate, risk, endpoint health, authorization, step-up, revocation, and recovery. The design should name its authoritative inputs, state transitions, dependencies, limits, security boundaries, operational owner, telemetry, failure behavior, and acceptance criteria.

This guide traces the subject through the peer reachability, IKE exchange, authentication, child SA, selector, route, NAT, encapsulation, endpoint, and application chain. A configured feature, established session, successful control-plane exchange, or green dashboard is not sufficient proof. Intended state must be reconciled with applied state, real traffic or transactions, negative tests, degraded behavior, recovery, and the resulting service or security outcome.

A production implementation starts with a minimal known-good baseline and explicit invariants. Introduce one dependency or capability at a time, preserve before-state evidence, test positive and prohibited behavior, inject realistic failures, measure convergence or recovery, and retain a tested rollback. Automation must stop safely when authority, freshness, identity, capacity, or observed outcomes are ambiguous.

Troubleshooting locates the first divergence from the expected state machine or end-to-end transaction. Inspect both directions and all administrative boundaries; account for caching, eventual consistency, stale identity, time, recursive dependencies, asymmetric paths, encapsulation, hardware offload, partial failure, and missing telemetry. Closure requires a causal explanation, reproducible evidence, validated restoration, residual limitations, and prevention or monitoring actions.

Engineering practices

Define authority, ownership, scope, dependencies, limits, and acceptance criteria for identity, mfa, posture, and device trust.

Model the expected state, trust boundaries, and transaction before implementation or troubleshooting.

Validate intended configuration, active control state, applied enforcement or forwarding, and end-to-end outcome independently.

Test normal, prohibited, degraded, failed, recovery, upgrade, and rollback conditions.

Preserve topology, identity, policy, configuration, packet, telemetry, log, timing, and service evidence.

Automate through typed data, deterministic gates, canaries, bounded authority, and reversible changes.

Failure patterns

Treating identity, mfa, posture, and device trust as a vendor feature instead of an end-to-end engineering system.

Relying on intended configuration or controller state without proving applied and operational behavior.

Ignoring reverse paths, external dependencies, identity, time, resource limits, or partial failure.

Changing several variables before preserving evidence and stating a falsifiable hypothesis.

Allowing automation or AI to act on stale, conflicting, unverified, or unauthorized inputs.

Declaring success after one positive probe without negative tests, scale checks, rollback, and recurrence monitoring.

Validation and evidence

Method	Expected evidence
Examine	Requirements, architecture, identities, models, policy, configuration, ownership, and dependencies for identity, mfa, posture, and device trust.
Observe	Control state, applied policy or forwarding, telemetry, logs, packets, sessions, resources, and endpoint behavior.
Test	Expected behavior, prohibited behavior, dependency loss, partial failure, scale, recovery, and rollback.
Measure	Availability, latency, loss, convergence, risk, resource use, change success, detection quality, and user or mission outcome.
Reconcile	Intended state against observed network, security, identity, application, data, and evidence records.

Chapter 05 laboratory and review

Practical laboratory

Model a remote-access service with certificate and MFA, posture, split and full tunnel, internal DNS, and application access.

Laboratory evidence package

- Architecture or topology showing boundaries, identities, dependencies, and authoritative inputs.
- Versioned configuration or code with explanatory comments and reproducible setup instructions.
- Nominal-path validation plus at least two injected failure or boundary conditions.
- Structured logs, traces, metrics, packet captures, test output, or other appropriate evidence.
- A concise decision record covering findings, limitations, residual risks, and next actions.

Frontier extension

Explore passkey-backed access, device attestation, continuous authorization, browser isolation, and agent identity.

Review gate

Advance only when another engineer can reproduce the result, explain the architecture, identify the failure boundary, and distinguish measured evidence from assumptions or projections.

CHAPTER 06

MTU, fragmentation, performance, and mobility

Maintain service when encapsulation, network changes, and large handshakes alter packets.

Chapter operating position

Maintain service when encapsulation, network changes, and large handshakes alter packets.

6.1 Encapsulation overhead and PMTUD

Encapsulation overhead and PMTUD must be engineered as an observable mechanism rather than a product label or checklist item. Calculate ESP, UDP, tunnel, TLS, and inner overhead; manage MSS, ICMP, IPv6, and black holes. The design should name its authoritative inputs, state transitions, dependencies, limits, security boundaries, operational owner, telemetry, failure behavior, and acceptance criteria.

This guide traces the subject through the peer reachability, IKE exchange, authentication, child SA, selector, route, NAT, encapsulation, endpoint, and application chain. A configured feature, established session, successful control-plane exchange, or green dashboard is not sufficient proof. Intended state must be reconciled with applied state, real traffic or transactions, negative tests, degraded behavior, recovery, and the resulting service or security outcome.

A production implementation starts with a minimal known-good baseline and explicit invariants. Introduce one dependency or capability at a time, preserve before-state evidence, test positive and prohibited behavior, inject realistic failures, measure convergence or recovery, and retain a tested rollback. Automation must stop safely when authority, freshness, identity, capacity, or observed outcomes are ambiguous.

Troubleshooting locates the first divergence from the expected state machine or end-to-end transaction. Inspect both directions and all administrative boundaries; account for caching, eventual consistency, stale identity, time, recursive dependencies, asymmetric paths, encapsulation, hardware offload, partial failure, and missing telemetry. Closure requires a causal explanation, reproducible evidence, validated restoration, residual limitations, and prevention or monitoring actions.

Engineering practices

- Define authority, ownership, scope, dependencies, limits, and acceptance criteria for encapsulation overhead and pmtud.
- Model the expected state, trust boundaries, and transaction before implementation or troubleshooting.
- Validate intended configuration, active control state, applied enforcement or forwarding, and end-to-end outcome independently.
- Test normal, prohibited, degraded, failed, recovery, upgrade, and rollback conditions.
- Preserve topology, identity, policy, configuration, packet, telemetry, log, timing, and service evidence.
- Automate through typed data, deterministic gates, canaries, bounded authority, and reversible changes.

Failure patterns

- Treating encapsulation overhead and pmtud as a vendor feature instead of an end-to-end engineering system.
- Relying on intended configuration or controller state without proving applied and operational behavior.
- Ignoring reverse paths, external dependencies, identity, time, resource limits, or partial failure.
- Changing several variables before preserving evidence and stating a falsifiable hypothesis.
- Allowing automation or AI to act on stale, conflicting, unverified, or unauthorized inputs.
- Declaring success after one positive probe without negative tests, scale checks, rollback, and recurrence monitoring.

Validation and evidence

Method	Expected evidence
Examine	Requirements, architecture, identities, models, policy, configuration, ownership, and dependencies for encapsulation overhead and pmtud.
Observe	Control state, applied policy or forwarding, telemetry, logs, packets, sessions, resources, and endpoint behavior.
Test	Expected behavior, prohibited behavior, dependency loss, partial failure, scale, recovery, and rollback.
Measure	Availability, latency, loss, convergence, risk, resource use, change success, detection quality, and user or mission outcome.
Reconcile	Intended state against observed network, security, identity, application, data, and evidence records.

6.2 IKE fragmentation and large messages

IKE fragmentation and large messages must be engineered as an observable mechanism rather than a product label or checklist item. Use fragmentation and intermediate exchanges with certificates, multiple exchanges, and middleboxes. The design should name its authoritative inputs, state transitions, dependencies, limits, security boundaries, operational owner, telemetry, failure behavior, and acceptance criteria.

This guide traces the subject through the peer reachability, IKE exchange, authentication, child SA, selector, route, NAT, encapsulation, endpoint, and application chain. A configured feature, established session, successful control-plane exchange, or green dashboard is not sufficient proof. Intended state must be reconciled with applied state, real traffic or transactions, negative tests, degraded behavior, recovery, and the resulting service or security outcome.

A production implementation starts with a minimal known-good baseline and explicit invariants. Introduce one dependency or capability at a time, preserve before-state evidence, test positive and prohibited behavior, inject realistic failures, measure convergence or recovery, and retain a tested rollback. Automation must stop safely when authority, freshness, identity, capacity, or observed outcomes are ambiguous.

Troubleshooting locates the first divergence from the expected state machine or end-to-end transaction. Inspect both directions and all administrative boundaries; account for caching, eventual consistency, stale identity, time, recursive dependencies, asymmetric paths, encapsulation, hardware offload, partial failure, and missing telemetry. Closure requires a causal explanation, reproducible evidence, validated restoration, residual limitations, and prevention or monitoring actions.

Engineering practices

- Define authority, ownership, scope, dependencies, limits, and acceptance criteria for ike fragmentation and large messages.
- Model the expected state, trust boundaries, and transaction before implementation or troubleshooting.
- Validate intended configuration, active control state, applied enforcement or forwarding, and end-to-end outcome independently.
- Test normal, prohibited, degraded, failed, recovery, upgrade, and rollback conditions.
- Preserve topology, identity, policy, configuration, packet, telemetry, log, timing, and service evidence.
- Automate through typed data, deterministic gates, canaries, bounded authority, and reversible changes.

Implementation sketch

```
ikev2_profile:
  authentication: certificate
  proposals:
    encryption: aes-gcm-256
    prf: hmac-sha2-384
    key_exchange: ecp384
  transition:
    multiple_key_exchange: evaluate
    ppk: controlled-pilot
    fragmentation: enabled
  child_sa:
    local: 10.10.0.0/16
    remote: 10.20.0.0/16
  validation:
    - ike_auth_identity
    - selectors_and_routes
    - nat_and_mtu
    - rekey_and_failover
```

Failure patterns

Treating ike fragmentation and large messages as a vendor feature instead of an end-to-end engineering system.

Relying on intended configuration or controller state without proving applied and operational behavior.

Ignoring reverse paths, external dependencies, identity, time, resource limits, or partial failure.

Changing several variables before preserving evidence and stating a falsifiable hypothesis.

Allowing automation or AI to act on stale, conflicting, unverified, or unauthorized inputs.

Declaring success after one positive probe without negative tests, scale checks, rollback, and recurrence monitoring.

Validation and evidence

Method	Expected evidence
Examine	Requirements, architecture, identities, models, policy, configuration, ownership, and dependencies for ike fragmentation and large messages.
Observe	Control state, applied policy or forwarding, telemetry, logs, packets, sessions, resources, and endpoint behavior.
Test	Expected behavior, prohibited behavior, dependency loss, partial failure, scale, recovery, and rollback.
Measure	Availability, latency, loss, convergence, risk, resource use, change success, detection quality, and user or mission outcome.
Reconcile	Intended state against observed network, security, identity, application, data, and evidence records.

6.3 Throughput and acceleration

Throughput and acceleration must be engineered as an observable mechanism rather than a product label or checklist item. Measure crypto, packets, sessions, latency, CPU, hardware offload, small packets, rekey, and inspection. The design should name its authoritative inputs, state transitions, dependencies, limits, security boundaries, operational owner, telemetry, failure behavior, and acceptance criteria.

This guide traces the subject through the peer reachability, IKE exchange, authentication, child SA, selector, route, NAT, encapsulation, endpoint, and application chain. A configured feature, established session, successful control-plane exchange, or green dashboard is not sufficient proof. Intended state must be reconciled with applied state, real traffic or transactions, negative tests, degraded behavior, recovery, and the resulting service or security outcome.

A production implementation starts with a minimal known-good baseline and explicit invariants. Introduce one dependency or capability at a time, preserve before-state evidence, test positive and prohibited behavior, inject realistic failures, measure convergence or recovery, and retain a tested rollback. Automation must stop safely when authority, freshness, identity, capacity, or observed outcomes are ambiguous.

Troubleshooting locates the first divergence from the expected state machine or end-to-end transaction. Inspect both directions and all administrative boundaries; account for caching, eventual consistency, stale identity, time, recursive dependencies,

asymmetric paths, encapsulation, hardware offload, partial failure, and missing telemetry. Closure requires a causal explanation, reproducible evidence, validated restoration, residual limitations, and prevention or monitoring actions.

Engineering practices

Define authority, ownership, scope, dependencies, limits, and acceptance criteria for throughput and acceleration.

Model the expected state, trust boundaries, and transaction before implementation or troubleshooting.

Validate intended configuration, active control state, applied enforcement or forwarding, and end-to-end outcome independently.

Test normal, prohibited, degraded, failed, recovery, upgrade, and rollback conditions.

Preserve topology, identity, policy, configuration, packet, telemetry, log, timing, and service evidence.

Automate through typed data, deterministic gates, canaries, bounded authority, and reversible changes.

Failure patterns

Treating throughput and acceleration as a vendor feature instead of an end-to-end engineering system.

Relying on intended configuration or controller state without proving applied and operational behavior.

Ignoring reverse paths, external dependencies, identity, time, resource limits, or partial failure.

Changing several variables before preserving evidence and stating a falsifiable hypothesis.

Allowing automation or AI to act on stale, conflicting, unverified, or unauthorized inputs.

Declaring success after one positive probe without negative tests, scale checks, rollback, and recurrence monitoring.

Validation and evidence

Method	Expected evidence
Examine	Requirements, architecture, identities, models, policy, configuration, ownership, and dependencies for throughput and acceleration.
Observe	Control state, applied policy or forwarding, telemetry, logs, packets, sessions, resources, and endpoint behavior.
Test	Expected behavior, prohibited behavior, dependency loss, partial failure, scale, recovery, and rollback.
Measure	Availability, latency, loss, convergence, risk, resource use, change success, detection quality, and user or mission outcome.
Reconcile	Intended state against observed network, security, identity, application, data, and evidence records.

6.4 Mobility and multihoming

Mobility and multihoming must be engineered as an observable mechanism rather than a product label or checklist item. Handle client network changes, clusters, multiple interfaces, reauthentication, state, and continuity. The design should name its authoritative inputs, state transitions, dependencies, limits, security boundaries, operational owner, telemetry, failure behavior, and acceptance criteria.

This guide traces the subject through the peer reachability, IKE exchange, authentication, child SA, selector, route, NAT, encapsulation, endpoint, and application chain. A configured feature, established session, successful control-plane exchange, or green dashboard is not sufficient proof. Intended state must be reconciled with applied state, real traffic or transactions, negative tests, degraded behavior, recovery, and the resulting service or security outcome.

A production implementation starts with a minimal known-good baseline and explicit invariants. Introduce one dependency or capability at a time, preserve before-state evidence, test positive and prohibited behavior, inject realistic failures, measure convergence or recovery, and retain a tested rollback. Automation must stop safely when authority, freshness, identity, capacity, or observed outcomes are ambiguous.

Troubleshooting locates the first divergence from the expected state machine or end-to-end transaction. Inspect both directions and all administrative boundaries; account for caching, eventual consistency, stale identity, time, recursive dependencies, asymmetric paths, encapsulation, hardware offload, partial failure, and missing telemetry. Closure requires a causal explanation, reproducible evidence, validated restoration, residual limitations, and prevention or monitoring actions.

Engineering practices

Define authority, ownership, scope, dependencies, limits, and acceptance criteria for mobility and multihoming.

Model the expected state, trust boundaries, and transaction before implementation or troubleshooting.

Validate intended configuration, active control state, applied enforcement or forwarding, and end-to-end outcome independently.

Test normal, prohibited, degraded, failed, recovery, upgrade, and rollback conditions.

Preserve topology, identity, policy, configuration, packet, telemetry, log, timing, and service evidence.

Automate through typed data, deterministic gates, canaries, bounded authority, and reversible changes.

Failure patterns

Treating mobility and multihoming as a vendor feature instead of an end-to-end engineering system.

Relying on intended configuration or controller state without proving applied and operational behavior.

Ignoring reverse paths, external dependencies, identity, time, resource limits, or partial failure.

Changing several variables before preserving evidence and stating a falsifiable hypothesis.

Allowing automation or AI to act on stale, conflicting, unverified, or unauthorized inputs.

Declaring success after one positive probe without negative tests, scale checks, rollback, and recurrence monitoring.

Validation and evidence

Method	Expected evidence
Examine	Requirements, architecture, identities, models, policy, configuration, ownership, and dependencies for mobility and multihoming.
Observe	Control state, applied policy or forwarding, telemetry, logs, packets, sessions, resources, and endpoint behavior.
Test	Expected behavior, prohibited behavior, dependency loss, partial failure, scale, recovery, and rollback.
Measure	Availability, latency, loss, convergence, risk, resource use, change success, detection quality, and user or mission outcome.
Reconcile	Intended state against observed network, security, identity, application, data, and evidence records.

Chapter 06 laboratory and review

Practical laboratory

Create an MTU black hole, large-certificate exchange, path change, and high-throughput load; measure negotiation and impact.

Laboratory evidence package

- Architecture or topology showing boundaries, identities, dependencies, and authoritative inputs.
- Versioned configuration or code with explanatory comments and reproducible setup instructions.
- Nominal-path validation plus at least two injected failure or boundary conditions.
- Structured logs, traces, metrics, packet captures, test output, or other appropriate evidence.
- A concise decision record covering findings, limitations, residual risks, and next actions.

Frontier extension

Evaluate PQC sizing, QUIC access research, multipath tunnels, DPU crypto, and satellite behavior.

Review gate

Advance only when another engineer can reproduce the result, explain the architecture, identify the failure boundary, and distinguish measured evidence from assumptions or projections.

CHAPTER 07

High availability, operations, and security

Operate VPN services as critical identity and routing systems.

Chapter operating position
Operate VPN services as critical identity and routing systems.

7.1 Cluster state and peer behavior

Cluster state and peer behavior must be engineered as an observable mechanism rather than a product label or checklist item. Understand active/passive or active/active state, SA synchronization, peer identity, routing, continuity, and split brain. The design should name its authoritative inputs, state transitions, dependencies, limits, security boundaries, operational owner, telemetry, failure behavior, and acceptance criteria.

This guide traces the subject through the peer reachability, IKE exchange, authentication, child SA, selector, route, NAT, encapsulation, endpoint, and application chain. A configured feature, established session, successful control-plane exchange, or green dashboard is not sufficient proof. Intended state must be reconciled with applied state, real traffic or transactions, negative tests, degraded behavior, recovery, and the resulting service or security outcome.

A production implementation starts with a minimal known-good baseline and explicit invariants. Introduce one dependency or capability at a time, preserve before-state evidence, test positive and prohibited behavior, inject realistic failures, measure convergence or recovery, and retain a tested rollback. Automation must stop safely when authority, freshness, identity, capacity, or observed outcomes are ambiguous.

Troubleshooting locates the first divergence from the expected state machine or end-to-end transaction. Inspect both directions and all administrative boundaries; account for caching, eventual consistency, stale identity, time, recursive dependencies, asymmetric paths, encapsulation, hardware offload, partial failure, and missing telemetry. Closure requires a causal explanation, reproducible evidence, validated restoration, residual limitations, and prevention or monitoring actions.

Engineering practices

- Define authority, ownership, scope, dependencies, limits, and acceptance criteria for cluster state and peer behavior.
- Model the expected state, trust boundaries, and transaction before implementation or troubleshooting.
- Validate intended configuration, active control state, applied enforcement or forwarding, and end-to-end outcome independently.
- Test normal, prohibited, degraded, failed, recovery, upgrade, and rollback conditions.
- Preserve topology, identity, policy, configuration, packet, telemetry, log, timing, and service evidence.
- Automate through typed data, deterministic gates, canaries, bounded authority, and reversible changes.

Failure patterns

- Treating cluster state and peer behavior as a vendor feature instead of an end-to-end engineering system.
- Relying on intended configuration or controller state without proving applied and operational behavior.
- Ignoring reverse paths, external dependencies, identity, time, resource limits, or partial failure.
- Changing several variables before preserving evidence and stating a falsifiable hypothesis.
- Allowing automation or AI to act on stale, conflicting, unverified, or unauthorized inputs.
- Declaring success after one positive probe without negative tests, scale checks, rollback, and recurrence monitoring.

Validation and evidence

Method	Expected evidence
Examine	Requirements, architecture, identities, models, policy, configuration, ownership, and dependencies for cluster state and peer behavior.
Observe	Control state, applied policy or forwarding, telemetry, logs, packets, sessions, resources, and endpoint behavior.
Test	Expected behavior, prohibited behavior, dependency loss, partial failure, scale, recovery, and rollback.
Measure	Availability, latency, loss, convergence, risk, resource use, change success, detection quality, and user or mission outcome.
Reconcile	Intended state against observed network, security, identity, application, data, and evidence records.

7.2 Logging, telemetry, and privacy

Logging, telemetry, and privacy must be engineered as an observable mechanism rather than a product label or checklist item. Collect negotiation, identity, certificate, selector, packet, performance, route, posture, and application evidence. The design should name its authoritative inputs, state transitions, dependencies, limits, security boundaries, operational owner, telemetry, failure behavior, and acceptance criteria.

This guide traces the subject through the peer reachability, IKE exchange, authentication, child SA, selector, route, NAT, encapsulation, endpoint, and application chain. A configured feature, established session, successful control-plane exchange, or green dashboard is not sufficient proof. Intended state must be reconciled with applied state, real traffic or transactions, negative tests, degraded behavior, recovery, and the resulting service or security outcome.

A production implementation starts with a minimal known-good baseline and explicit invariants. Introduce one dependency or capability at a time, preserve before-state evidence, test positive and prohibited behavior, inject realistic failures, measure convergence or recovery, and retain a tested rollback. Automation must stop safely when authority, freshness, identity, capacity, or observed outcomes are ambiguous.

Troubleshooting locates the first divergence from the expected state machine or end-to-end transaction. Inspect both directions and all administrative boundaries; account for caching, eventual consistency, stale identity, time, recursive dependencies, asymmetric paths, encapsulation, hardware offload, partial failure, and missing telemetry. Closure requires a causal explanation, reproducible evidence, validated restoration, residual limitations, and prevention or monitoring actions.

Engineering practices

- Define authority, ownership, scope, dependencies, limits, and acceptance criteria for logging, telemetry, and privacy.
- Model the expected state, trust boundaries, and transaction before implementation or troubleshooting.
- Validate intended configuration, active control state, applied enforcement or forwarding, and end-to-end outcome independently.
- Test normal, prohibited, degraded, failed, recovery, upgrade, and rollback conditions.
- Preserve topology, identity, policy, configuration, packet, telemetry, log, timing, and service evidence.
- Automate through typed data, deterministic gates, canaries, bounded authority, and reversible changes.

Implementation sketch

```
ikev2_profile:
  authentication: certificate
  proposals:
    encryption: aes-gcm-256
    prf: hmac-sha2-384
    key_exchange: ecp384
  transition:
    multiple_key_exchange: evaluate
    ppk: controlled-pilot
    fragmentation: enabled
  child_sa:
    local: 10.10.0.0/16
    remote: 10.20.0.0/16
  validation:
    - ike_auth_identity
    - selectors_and_routes
    - nat_and_mtu
    - rekey_and_failover
```

Failure patterns

Treating logging, telemetry, and privacy as a vendor feature instead of an end-to-end engineering system.

Relying on intended configuration or controller state without proving applied and operational behavior.

Ignoring reverse paths, external dependencies, identity, time, resource limits, or partial failure.

Changing several variables before preserving evidence and stating a falsifiable hypothesis.

Allowing automation or AI to act on stale, conflicting, unverified, or unauthorized inputs.

Declaring success after one positive probe without negative tests, scale checks, rollback, and recurrence monitoring.

Validation and evidence

Method	Expected evidence
Examine	Requirements, architecture, identities, models, policy, configuration, ownership, and dependencies for logging, telemetry, and privacy.
Observe	Control state, applied policy or forwarding, telemetry, logs, packets, sessions, resources, and endpoint behavior.
Test	Expected behavior, prohibited behavior, dependency loss, partial failure, scale, recovery, and rollback.
Measure	Availability, latency, loss, convergence, risk, resource use, change success, detection quality, and user or mission outcome.
Reconcile	Intended state against observed network, security, identity, application, data, and evidence records.

7.3 Key and credential lifecycle

Key and credential lifecycle must be engineered as an observable mechanism rather than a product label or checklist item. Rotate certificates, PSKs, trust anchors, algorithms, clients, software, and revocation without uncontrolled outages. The design should name its authoritative inputs, state transitions, dependencies, limits, security boundaries, operational owner, telemetry, failure behavior, and acceptance criteria.

This guide traces the subject through the peer reachability, IKE exchange, authentication, child SA, selector, route, NAT, encapsulation, endpoint, and application chain. A configured feature, established session, successful control-plane exchange, or green dashboard is not sufficient proof. Intended state must be reconciled with applied state, real traffic or transactions, negative tests, degraded behavior, recovery, and the resulting service or security outcome.

A production implementation starts with a minimal known-good baseline and explicit invariants. Introduce one dependency or capability at a time, preserve before-state evidence, test positive and prohibited behavior, inject realistic failures, measure convergence or recovery, and retain a tested rollback. Automation must stop safely when authority, freshness, identity, capacity, or observed outcomes are ambiguous.

Troubleshooting locates the first divergence from the expected state machine or end-to-end transaction. Inspect both directions and all administrative boundaries; account for caching, eventual consistency, stale identity, time, recursive dependencies,

asymmetric paths, encapsulation, hardware offload, partial failure, and missing telemetry. Closure requires a causal explanation, reproducible evidence, validated restoration, residual limitations, and prevention or monitoring actions.

Engineering practices

Define authority, ownership, scope, dependencies, limits, and acceptance criteria for key and credential lifecycle.

Model the expected state, trust boundaries, and transaction before implementation or troubleshooting.

Validate intended configuration, active control state, applied enforcement or forwarding, and end-to-end outcome independently.

Test normal, prohibited, degraded, failed, recovery, upgrade, and rollback conditions.

Preserve topology, identity, policy, configuration, packet, telemetry, log, timing, and service evidence.

Automate through typed data, deterministic gates, canaries, bounded authority, and reversible changes.

Failure patterns

Treating key and credential lifecycle as a vendor feature instead of an end-to-end engineering system.

Relying on intended configuration or controller state without proving applied and operational behavior.

Ignoring reverse paths, external dependencies, identity, time, resource limits, or partial failure.

Changing several variables before preserving evidence and stating a falsifiable hypothesis.

Allowing automation or AI to act on stale, conflicting, unverified, or unauthorized inputs.

Declaring success after one positive probe without negative tests, scale checks, rollback, and recurrence monitoring.

Validation and evidence

Method	Expected evidence
Examine	Requirements, architecture, identities, models, policy, configuration, ownership, and dependencies for key and credential lifecycle.
Observe	Control state, applied policy or forwarding, telemetry, logs, packets, sessions, resources, and endpoint behavior.
Test	Expected behavior, prohibited behavior, dependency loss, partial failure, scale, recovery, and rollback.
Measure	Availability, latency, loss, convergence, risk, resource use, change success, detection quality, and user or mission outcome.
Reconcile	Intended state against observed network, security, identity, application, data, and evidence records.

7.4 Attack surface and hardening

Attack surface and hardening must be engineered as an observable mechanism rather than a product label or checklist item. Protect gateways, management, authentication, portals, clients, updates, credential stores, and DoS controls. The design should name its authoritative inputs, state transitions, dependencies, limits, security boundaries, operational owner, telemetry, failure behavior, and acceptance criteria.

This guide traces the subject through the peer reachability, IKE exchange, authentication, child SA, selector, route, NAT, encapsulation, endpoint, and application chain. A configured feature, established session, successful control-plane exchange, or green dashboard is not sufficient proof. Intended state must be reconciled with applied state, real traffic or transactions, negative tests, degraded behavior, recovery, and the resulting service or security outcome.

A production implementation starts with a minimal known-good baseline and explicit invariants. Introduce one dependency or capability at a time, preserve before-state evidence, test positive and prohibited behavior, inject realistic failures, measure convergence or recovery, and retain a tested rollback. Automation must stop safely when authority, freshness, identity, capacity, or observed outcomes are ambiguous.

Troubleshooting locates the first divergence from the expected state machine or end-to-end transaction. Inspect both directions and all administrative boundaries; account for caching, eventual consistency, stale identity, time, recursive dependencies, asymmetric paths, encapsulation, hardware offload, partial failure, and missing telemetry. Closure requires a causal explanation, reproducible evidence, validated restoration, residual limitations, and prevention or monitoring actions.

Engineering practices

Define authority, ownership, scope, dependencies, limits, and acceptance criteria for attack surface and hardening.

Model the expected state, trust boundaries, and transaction before implementation or troubleshooting.

Validate intended configuration, active control state, applied enforcement or forwarding, and end-to-end outcome independently.

Test normal, prohibited, degraded, failed, recovery, upgrade, and rollback conditions.

Preserve topology, identity, policy, configuration, packet, telemetry, log, timing, and service evidence.

Automate through typed data, deterministic gates, canaries, bounded authority, and reversible changes.

Failure patterns

Treating attack surface and hardening as a vendor feature instead of an end-to-end engineering system.

Relying on intended configuration or controller state without proving applied and operational behavior.

Ignoring reverse paths, external dependencies, identity, time, resource limits, or partial failure.

Changing several variables before preserving evidence and stating a falsifiable hypothesis.

Allowing automation or AI to act on stale, conflicting, unverified, or unauthorized inputs.

Declaring success after one positive probe without negative tests, scale checks, rollback, and recurrence monitoring.

Validation and evidence

Method	Expected evidence
Examine	Requirements, architecture, identities, models, policy, configuration, ownership, and dependencies for attack surface and hardening.
Observe	Control state, applied policy or forwarding, telemetry, logs, packets, sessions, resources, and endpoint behavior.
Test	Expected behavior, prohibited behavior, dependency loss, partial failure, scale, recovery, and rollback.
Measure	Availability, latency, loss, convergence, risk, resource use, change success, detection quality, and user or mission outcome.
Reconcile	Intended state against observed network, security, identity, application, data, and evidence records.

Chapter 07 laboratory and review

Practical laboratory
Run continuity tests covering node failure, certificate rollover, identity outage, route failure, client update failure, and revocation.

Laboratory evidence package

- Architecture or topology showing boundaries, identities, dependencies, and authoritative inputs.
- Versioned configuration or code with explanatory comments and reproducible setup instructions.
- Nominal-path validation plus at least two injected failure or boundary conditions.
- Structured logs, traces, metrics, packet captures, test output, or other appropriate evidence.
- A concise decision record covering findings, limitations, residual risks, and next actions.

Frontier extension

Explore isolated key services, confidential VPN processing, signed tunnel evidence, and autonomous containment.

Review gate
Advance only when another engineer can reproduce the result, explain the architecture, identify the failure boundary, and distinguish measured evidence from assumptions or projections.

CHAPTER 08

Tunnel troubleshooting and recovery

Use a phase-oriented workflow from reachability through negotiation, SAs, and application traffic.

Chapter operating position

Use a phase-oriented workflow from reachability through negotiation, SAs, and application traffic.

8.1 Peer and transport reachability

Peer and transport reachability must be engineered as an observable mechanism rather than a product label or checklist item. Verify addresses, routing, NAT, ports, UDP encapsulation, fragmentation, access lists, load balancers, and return. The design should name its authoritative inputs, state transitions, dependencies, limits, security boundaries, operational owner, telemetry, failure behavior, and acceptance criteria.

This guide traces the subject through the peer reachability, IKE exchange, authentication, child SA, selector, route, NAT, encapsulation, endpoint, and application chain. A configured feature, established session, successful control-plane exchange, or green dashboard is not sufficient proof. Intended state must be reconciled with applied state, real traffic or transactions, negative tests, degraded behavior, recovery, and the resulting service or security outcome.

A production implementation starts with a minimal known-good baseline and explicit invariants. Introduce one dependency or capability at a time, preserve before-state evidence, test positive and prohibited behavior, inject realistic failures, measure convergence or recovery, and retain a tested rollback. Automation must stop safely when authority, freshness, identity, capacity, or observed outcomes are ambiguous.

Troubleshooting locates the first divergence from the expected state machine or end-to-end transaction. Inspect both directions and all administrative boundaries; account for caching, eventual consistency, stale identity, time, recursive dependencies, asymmetric paths, encapsulation, hardware offload, partial failure, and missing telemetry. Closure requires a causal explanation, reproducible evidence, validated restoration, residual limitations, and prevention or monitoring actions.

Engineering practices

- Define authority, ownership, scope, dependencies, limits, and acceptance criteria for peer and transport reachability.
- Model the expected state, trust boundaries, and transaction before implementation or troubleshooting.
- Validate intended configuration, active control state, applied enforcement or forwarding, and end-to-end outcome independently.
- Test normal, prohibited, degraded, failed, recovery, upgrade, and rollback conditions.
- Preserve topology, identity, policy, configuration, packet, telemetry, log, timing, and service evidence.
- Automate through typed data, deterministic gates, canaries, bounded authority, and reversible changes.

Failure patterns

- Treating peer and transport reachability as a vendor feature instead of an end-to-end engineering system.
- Relying on intended configuration or controller state without proving applied and operational behavior.
- Ignoring reverse paths, external dependencies, identity, time, resource limits, or partial failure.
- Changing several variables before preserving evidence and stating a falsifiable hypothesis.
- Allowing automation or AI to act on stale, conflicting, unverified, or unauthorized inputs.
- Declaring success after one positive probe without negative tests, scale checks, rollback, and recurrence monitoring.

Validation and evidence

Method	Expected evidence
Examine	Requirements, architecture, identities, models, policy, configuration, ownership, and dependencies for peer and transport reachability.
Observe	Control state, applied policy or forwarding, telemetry, logs, packets, sessions, resources, and endpoint behavior.
Test	Expected behavior, prohibited behavior, dependency loss, partial failure, scale, recovery, and rollback.
Measure	Availability, latency, loss, convergence, risk, resource use, change success, detection quality, and user or mission outcome.
Reconcile	Intended state against observed network, security, identity, application, data, and evidence records.

8.2 Proposal, identity, and authentication

Proposal, identity, and authentication must be engineered as an observable mechanism rather than a product label or checklist item. Compare transforms, groups, IDs, certificates, trust, time, PSKs, MFA, EAP, and authorization. The design should name its authoritative inputs, state transitions, dependencies, limits, security boundaries, operational owner, telemetry, failure behavior, and acceptance criteria.

This guide traces the subject through the peer reachability, IKE exchange, authentication, child SA, selector, route, NAT, encapsulation, endpoint, and application chain. A configured feature, established session, successful control-plane exchange, or green dashboard is not sufficient proof. Intended state must be reconciled with applied state, real traffic or transactions, negative tests, degraded behavior, recovery, and the resulting service or security outcome.

A production implementation starts with a minimal known-good baseline and explicit invariants. Introduce one dependency or capability at a time, preserve before-state evidence, test positive and prohibited behavior, inject realistic failures, measure convergence or recovery, and retain a tested rollback. Automation must stop safely when authority, freshness, identity, capacity, or observed outcomes are ambiguous.

Troubleshooting locates the first divergence from the expected state machine or end-to-end transaction. Inspect both directions and all administrative boundaries; account for caching, eventual consistency, stale identity, time, recursive dependencies, asymmetric paths, encapsulation, hardware offload, partial failure, and missing telemetry. Closure requires a causal explanation, reproducible evidence, validated restoration, residual limitations, and prevention or monitoring actions.

Engineering practices

- Define authority, ownership, scope, dependencies, limits, and acceptance criteria for proposal, identity, and authentication.
- Model the expected state, trust boundaries, and transaction before implementation or troubleshooting.
- Validate intended configuration, active control state, applied enforcement or forwarding, and end-to-end outcome independently.
- Test normal, prohibited, degraded, failed, recovery, upgrade, and rollback conditions.
- Preserve topology, identity, policy, configuration, packet, telemetry, log, timing, and service evidence.
- Automate through typed data, deterministic gates, canaries, bounded authority, and reversible changes.

Implementation sketch

```
ikev2_profile:
  authentication: certificate
  proposals:
    encryption: aes-gcm-256
    prf: hmac-sha2-384
    key_exchange: ecp384
  transition:
    multiple_key_exchange: evaluate
    ppk: controlled-pilot
    fragmentation: enabled
  child_sa:
    local: 10.10.0.0/16
    remote: 10.20.0.0/16
  validation:
    - ike_auth_identity
    - selectors_and_routes
    - nat_and_mtu
    - rekey_and_failover
```

Failure patterns

Treating proposal, identity, and authentication as a vendor feature instead of an end-to-end engineering system.

Relying on intended configuration or controller state without proving applied and operational behavior.

Ignoring reverse paths, external dependencies, identity, time, resource limits, or partial failure.

Changing several variables before preserving evidence and stating a falsifiable hypothesis.

Allowing automation or AI to act on stale, conflicting, unverified, or unauthorized inputs.

Declaring success after one positive probe without negative tests, scale checks, rollback, and recurrence monitoring.

Validation and evidence

Method	Expected evidence
Examine	Requirements, architecture, identities, models, policy, configuration, ownership, and dependencies for proposal, identity, and authentication.
Observe	Control state, applied policy or forwarding, telemetry, logs, packets, sessions, resources, and endpoint behavior.
Test	Expected behavior, prohibited behavior, dependency loss, partial failure, scale, recovery, and rollback.
Measure	Availability, latency, loss, convergence, risk, resource use, change success, detection quality, and user or mission outcome.
Reconcile	Intended state against observed network, security, identity, application, data, and evidence records.

8.3 Child SA and traffic failure

Child SA and traffic failure must be engineered as an observable mechanism rather than a product label or checklist item. Check selectors, routes, NAT, policy, SPDs, SAs, counters, replay, MTU, and asymmetric state. The design should name its authoritative inputs, state transitions, dependencies, limits, security boundaries, operational owner, telemetry, failure behavior, and acceptance criteria.

This guide traces the subject through the peer reachability, IKE exchange, authentication, child SA, selector, route, NAT, encapsulation, endpoint, and application chain. A configured feature, established session, successful control-plane exchange, or green dashboard is not sufficient proof. Intended state must be reconciled with applied state, real traffic or transactions, negative tests, degraded behavior, recovery, and the resulting service or security outcome.

A production implementation starts with a minimal known-good baseline and explicit invariants. Introduce one dependency or capability at a time, preserve before-state evidence, test positive and prohibited behavior, inject realistic failures, measure convergence or recovery, and retain a tested rollback. Automation must stop safely when authority, freshness, identity, capacity, or observed outcomes are ambiguous.

Troubleshooting locates the first divergence from the expected state machine or end-to-end transaction. Inspect both directions and all administrative boundaries; account for caching, eventual consistency, stale identity, time, recursive dependencies,

asymmetric paths, encapsulation, hardware offload, partial failure, and missing telemetry. Closure requires a causal explanation, reproducible evidence, validated restoration, residual limitations, and prevention or monitoring actions.

Engineering practices

Define authority, ownership, scope, dependencies, limits, and acceptance criteria for child sa and traffic failure.

Model the expected state, trust boundaries, and transaction before implementation or troubleshooting.

Validate intended configuration, active control state, applied enforcement or forwarding, and end-to-end outcome independently.

Test normal, prohibited, degraded, failed, recovery, upgrade, and rollback conditions.

Preserve topology, identity, policy, configuration, packet, telemetry, log, timing, and service evidence.

Automate through typed data, deterministic gates, canaries, bounded authority, and reversible changes.

Failure patterns

Treating child sa and traffic failure as a vendor feature instead of an end-to-end engineering system.

Relying on intended configuration or controller state without proving applied and operational behavior.

Ignoring reverse paths, external dependencies, identity, time, resource limits, or partial failure.

Changing several variables before preserving evidence and stating a falsifiable hypothesis.

Allowing automation or AI to act on stale, conflicting, unverified, or unauthorized inputs.

Declaring success after one positive probe without negative tests, scale checks, rollback, and recurrence monitoring.

Validation and evidence

Method	Expected evidence
Examine	Requirements, architecture, identities, models, policy, configuration, ownership, and dependencies for child sa and traffic failure.
Observe	Control state, applied policy or forwarding, telemetry, logs, packets, sessions, resources, and endpoint behavior.
Test	Expected behavior, prohibited behavior, dependency loss, partial failure, scale, recovery, and rollback.
Measure	Availability, latency, loss, convergence, risk, resource use, change success, detection quality, and user or mission outcome.
Reconcile	Intended state against observed network, security, identity, application, data, and evidence records.

8.4 Client and application troubleshooting

Client and application troubleshooting must be engineered as an observable mechanism rather than a product label or checklist item. Inspect posture, DNS, routes, proxy, local firewall, split tunnel, certificates, application policy, and captures. The design should name its authoritative inputs, state transitions, dependencies, limits, security boundaries, operational owner, telemetry, failure behavior, and acceptance criteria.

This guide traces the subject through the peer reachability, IKE exchange, authentication, child SA, selector, route, NAT, encapsulation, endpoint, and application chain. A configured feature, established session, successful control-plane exchange, or green dashboard is not sufficient proof. Intended state must be reconciled with applied state, real traffic or transactions, negative tests, degraded behavior, recovery, and the resulting service or security outcome.

A production implementation starts with a minimal known-good baseline and explicit invariants. Introduce one dependency or capability at a time, preserve before-state evidence, test positive and prohibited behavior, inject realistic failures, measure convergence or recovery, and retain a tested rollback. Automation must stop safely when authority, freshness, identity, capacity, or observed outcomes are ambiguous.

Troubleshooting locates the first divergence from the expected state machine or end-to-end transaction. Inspect both directions and all administrative boundaries; account for caching, eventual consistency, stale identity, time, recursive dependencies, asymmetric paths, encapsulation, hardware offload, partial failure, and missing telemetry. Closure requires a causal explanation, reproducible evidence, validated restoration, residual limitations, and prevention or monitoring actions.

Engineering practices

Define authority, ownership, scope, dependencies, limits, and acceptance criteria for client and application troubleshooting.

Model the expected state, trust boundaries, and transaction before implementation or troubleshooting.

Validate intended configuration, active control state, applied enforcement or forwarding, and end-to-end outcome independently.

Test normal, prohibited, degraded, failed, recovery, upgrade, and rollback conditions.

Preserve topology, identity, policy, configuration, packet, telemetry, log, timing, and service evidence.

Automate through typed data, deterministic gates, canaries, bounded authority, and reversible changes.

Failure patterns

Treating client and application troubleshooting as a vendor feature instead of an end-to-end engineering system.

Relying on intended configuration or controller state without proving applied and operational behavior.

Ignoring reverse paths, external dependencies, identity, time, resource limits, or partial failure.

Changing several variables before preserving evidence and stating a falsifiable hypothesis.

Allowing automation or AI to act on stale, conflicting, unverified, or unauthorized inputs.

Declaring success after one positive probe without negative tests, scale checks, rollback, and recurrence monitoring.

Validation and evidence

Method	Expected evidence
Examine	Requirements, architecture, identities, models, policy, configuration, ownership, and dependencies for client and application troubleshooting.
Observe	Control state, applied policy or forwarding, telemetry, logs, packets, sessions, resources, and endpoint behavior.
Test	Expected behavior, prohibited behavior, dependency loss, partial failure, scale, recovery, and rollback.
Measure	Availability, latency, loss, convergence, risk, resource use, change success, detection quality, and user or mission outcome.
Reconcile	Intended state against observed network, security, identity, application, data, and evidence records.

Chapter 08 laboratory and review

Practical laboratory
Troubleshoot hidden failures across IKE, authentication, selectors, route/NAT, MTU, and endpoint layers; produce a packet-backed timeline.

Laboratory evidence package

Architecture or topology showing boundaries, identities, dependencies, and authoritative inputs.

Versioned configuration or code with explanatory comments and reproducible setup instructions.

Nominal-path validation plus at least two injected failure or boundary conditions.

Structured logs, traces, metrics, packet captures, test output, or other appropriate evidence.

A concise decision record covering findings, limitations, residual risks, and next actions.

Frontier extension

Develop a governed tunnel analyst that recommends tests and stages rollback but cannot change production without approval.

Review gate
Advance only when another engineer can reproduce the result, explain the architecture, identify the failure boundary, and distinguish measured evidence from assumptions or projections.

Integrated learning roadmap

The guide is intended to be revisited. First establish fluency, then build a small implementation, operate it under failure, automate repeatable work, and finally evaluate emerging techniques against a stable baseline. Progress is demonstrated through evidence rather than reading completion alone.

Stage	Demonstrated capability
Foundation	Explain the system from first principles and trace its critical path without relying on product terminology.
Build	Implement the smallest representative system with explicit contracts, identities, telemetry, and tests.
Operate	Diagnose injected failures, recover safely, and preserve evidence of the causal chain.
Automate	Convert a proven manual workflow into bounded, idempotent, observable automation.
Architect	Design for scale, resilience, security, interoperability, and lifecycle change.
Explore	Evaluate frontier techniques using stated hypotheses, limitations, and adoption gates.

Source authority register

> Source policy
> Official standards, specifications, and project documentation remain with their issuing organizations. Mythos Systems provides original engineering interpretation, synthesis, implementation guidance, and relationship mapping. Source verification dates do not guarantee that an authority has not changed since review.

01. IETF / RFC Editor - RFC 4301 - Security Architecture for the Internet Protocol
Role: Base IPsec architecture, policy databases, and security associations.
Verified: 2026-07-26
Official source: https://www.rfc-editor.org/info/rfc4301

02. IETF / RFC Editor - RFC 7296 - Internet Key Exchange Protocol Version 2
Role: Internet Standard for IKEv2 mutual authentication and SA establishment.
Verified: 2026-07-26
Official source: https://www.rfc-editor.org/info/rfc7296

03. IETF / RFC Editor - RFC 8446 - TLS 1.3
Role: TLS 1.3 protocol specification.
Verified: 2026-07-26
Official source: https://www.rfc-editor.org/info/rfc8446

04. IETF / RFC Editor - RFC 9370 - Multiple Key Exchanges in IKEv2
Role: Enables multiple key exchanges for hybrid and transition use cases.
Verified: 2026-07-26
Official source: https://www.rfc-editor.org/info/rfc9370

05. IETF / RFC Editor - RFC 9242 - Intermediate Exchange in IKEv2
Role: Adds an exchange suitable for carrying larger key-establishment material.
Verified: 2026-07-26
Official source: https://www.rfc-editor.org/info/rfc9242

06. IETF / RFC Editor - RFC 7383 - IKEv2 Message Fragmentation
Role: Defines IKEv2 fragmentation for large messages.
Verified: 2026-07-26
Official source: https://www.rfc-editor.org/info/rfc7383

07. IETF / RFC Editor - RFC 9867 - Mixing Preshared Keys in IKEv2 for Post-Quantum Security
Role: Defines PPK mixing for quantum-resistant transition protection.
Verified: 2026-07-26
Official source: https://www.rfc-editor.org/info/rfc9867

08. NIST - SP 800-57 Part 1 Rev. 5 - Recommendation for Key Management
Role: Final key-management principles and lifecycle guidance.
Verified: 2026-07-26
Official source: https://csrc.nist.gov/pubs/sp/800/57/pt1/r5/final

Authority application and refresh triggers

Authority class	Required library action
Protocols and specifications	Recheck interoperability, security considerations, defaults, and migration guidance when a referenced specification changes.
Security and assurance frameworks	Update control mappings, evidence expectations, risk language, and assessment procedures after material framework revision.
Languages, runtimes, and projects	Re-run examples, tests, deprecation checks, and operational recommendations against supported releases.
Benchmarks and evaluations	Re-execute claims when workloads, methods, models, compilers, drivers, hardware, or scoring rules change.
Operational evidence	Incorporate validated incidents, failure tests, reader corrections, and measured implementation outcomes into the next revision.

Limitations, freshness, and revision control

Boundary	Statement
Publication limitation	This guide synthesizes broad engineering practice. It does not replace product documentation, legal advice, safety certification, or environment-specific design review.
Evidence limitation	Not every pattern is benchmarked in every environment. Examples illustrate engineering methods and must be validated against actual versions, workloads, hardware, policy, and failure conditions.
Freshness limitation	Vendor behavior, software libraries, AI models, security threats, and emerging standards can change quickly. Volatile claims require source revalidation before operational adoption.
Adoption limitation	Frontier content is not an implicit adoption recommendation. Adoption requires defined value, qualification gates, controlled proof, rollback, ownership, and residual-risk acceptance.
Status boundary	Candidate Focused Field Guide describes the publication, not certification of a product, implementation, engineer, or organization.

Revision record

Version	Revision statement
1.0.0-candidate	Initial candidate living-guide edition. Published 2026-07-26; scheduled review 2026-10-26.

Search and relationship metadata

Dimension	Engineering position
Primary domain	MYTHOS-SEC - Cybersecurity, Security Engineering & Cryptography
Secondary domain	MYTHOS-NET; MYTHOS-QTC; MYTHOS-CLD
Publication class	Field Guide / Canonical Living Overview Guide
Skill levels	L1 Foundational; L2 Practitioner; L3 Advanced; L4 Frontier
Tags	IPsec, IKEv2, TLS VPN, remote access, ESP, certificates, NAT traversal, MTU, post-quantum, tunnel troubleshooting
Canonical route	/library/field-guides/mythos-fg-sec-0004/

Living publication
Corrections, expanded laboratories, improved diagrams, authority changes, and new engineering evidence should revise this permanent publication rather than create disconnected replacements.