CANONICAL LIVING OVERVIEW GUIDE

Confidential Computing, TEEs, Attestation, FHE, and ZKP

An advanced assurance and privacy-enhancing cryptography guide covering trusted execution environments, data-in-use protection, threat models, remote attestation, RATS, EAT, key release, confidential containers and AI, multi-party data collaboration, fully homomorphic encryption, zero-knowledge proofs, integration,

PERMANENT ID
MYTHOS-FG-SEC-0018

PUBLICATION CLASS
FIELD GUIDE

VERSION / STATUS
1.0.0-candidate / CANDIDATE

PERMANENT PUBLICATION IDENTITY

Confidential Computing, TEEs, Attestation, FHE, and ZKP

Document ID
MYTHOS-FG-SEC-0018

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-EMT; MYTHOS-AI; MYTHOS-CLD
Audience	Security, cryptography, cloud, AI, platform, data, privacy, architecture, hardware, and research engineering teams.
Prerequisites	Applied cryptography, cloud and workload architecture, PKI, identity, key management, software supply chains, and basic algebraic concepts.
Publisher / owner	Mythos Systems / Aaron Stovall
Public classification	Public technical guidance

Expected outcomes

Explain what TEEs and confidential computing protect and what remains in the trusted computing base.

Design remote-attestation and key-release flows with evidence, reference values, appraisal, and policy.

Deploy confidential workloads with verified images, identities, secrets, data, updates, and recovery.

Evaluate FHE and ZKP according to concrete security, correctness, performance, and trust assumptions.

Choose among TEEs, FHE, ZKP, MPC, traditional encryption, and architectural controls without overclaiming.

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

Confidential computing and TEE foundations

Define data-in-use protection, isolation boundaries, and the trusted computing base.

Chapter operating position
Define data-in-use protection, isolation boundaries, and the trusted computing base.

1.1 Data at rest, in transit, and in use

Data at rest, in transit, and in use must be implemented as an observable engineering mechanism, not a product label or maturity claim. Separate storage encryption, secure channels, active computation, memory, registers, caches, I/O, and administrative access. The design should name authoritative inputs, identities, state transitions, dependencies, cryptographic or policy boundaries, limits, owners, telemetry, failure behavior, and acceptance criteria.

This guide traces the subject through the workload, TEE, evidence, verifier, appraisal, key release, protected computation, output, and relying-party decision chain. A configured integration, successful API response, established cryptographic session, green dashboard, or closed ticket is not sufficient proof. Intended state must be reconciled with applied state, negative tests, degraded behavior, raw evidence, recovery, and the resulting security or mission 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 authorized and prohibited behavior, inject realistic failures, measure convergence or recovery, and retain a tested rollback. Automation must stop safely when authority, freshness, identity, integrity, cryptographic state, or observed outcomes are ambiguous.

Troubleshooting locates the first divergence from the expected state machine or end-to-end transaction. Inspect every authority boundary and account for time, caching, eventual consistency, stale identity, incomplete collection, parser drift, asymmetric paths, encapsulation, supply-chain state, side channels, partial failure, and missing evidence. Closure requires causal explanation, reproducibility, validated restoration, residual limitations, and prevention or monitoring actions.

Engineering practices

- Define authority, ownership, scope, dependencies, limits, and acceptance criteria for data at rest, in transit, and in use.
- Model expected state, trust boundaries, data flow, identity, and failure behavior before implementation.
- Validate intended configuration, active state, applied enforcement, raw evidence, and end-to-end outcome independently.
- Test normal, prohibited, degraded, failed, scale, recovery, and rollback conditions.
- Preserve policy, source, version, identity, telemetry, artifact, packet, log, time, and decision evidence.
- Automate through typed contracts, least privilege, deterministic gates, canaries, bounded authority, and reversible changes.

Failure patterns

- Treating data at rest, in transit, and in use as a product feature instead of an end-to-end engineering system.
- Relying on intended policy, rule code, or controller state without proving applied and operational behavior.
- Ignoring data quality, identity, time, cryptographic lifecycle, supplier state, 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 result without negative tests, rollback, and recurrence monitoring.

Validation and evidence

Method	Expected evidence
Examine	Requirements, architecture, identities, data, algorithms, policy, configuration, ownership, and dependencies for data at rest, in transit, and in use.
Observe	Active state, applied policy, raw events, telemetry, artifacts, cryptographic results, resources, and endpoint behavior.
Test	Expected behavior, prohibited behavior, malformed inputs, dependency loss, partial failure, scale, recovery, and rollback.
Measure	Availability, latency, detection quality, evidence completeness, cryptographic cost, risk, resource use, and mission outcome.
Reconcile	Intended state against observed security, identity, application, cloud, cryptographic, data, and evidence records.

1.2 TEE architecture and isolation

TEE architecture and isolation must be implemented as an observable engineering mechanism, not a product label or maturity claim. Understand hardware roots, protected execution, memory encryption or integrity, privilege separation, VM or process models, and device support. The design should name authoritative inputs, identities, state transitions, dependencies, cryptographic or policy boundaries, limits, owners, telemetry, failure behavior, and acceptance criteria.

This guide traces the subject through the workload, TEE, evidence, verifier, appraisal, key release, protected computation, output, and relying-party decision chain. A configured integration, successful API response, established cryptographic session, green dashboard, or closed ticket is not sufficient proof. Intended state must be reconciled with applied state, negative tests, degraded behavior, raw evidence, recovery, and the resulting security or mission 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 authorized and prohibited behavior, inject realistic failures, measure convergence or recovery, and retain a tested rollback. Automation must stop safely when authority, freshness, identity, integrity, cryptographic state, or observed outcomes are ambiguous.

Troubleshooting locates the first divergence from the expected state machine or end-to-end transaction. Inspect every authority boundary and account for time, caching, eventual consistency, stale identity, incomplete collection, parser drift, asymmetric paths, encapsulation, supply-chain state, side channels, partial failure, and missing evidence. Closure requires causal explanation, reproducibility, validated restoration, residual limitations, and prevention or monitoring actions.

Engineering practices

- Define authority, ownership, scope, dependencies, limits, and acceptance criteria for tee architecture and isolation.
- Model expected state, trust boundaries, data flow, identity, and failure behavior before implementation.
- Validate intended configuration, active state, applied enforcement, raw evidence, and end-to-end outcome independently.
- Test normal, prohibited, degraded, failed, scale, recovery, and rollback conditions.
- Preserve policy, source, version, identity, telemetry, artifact, packet, log, time, and decision evidence.
- Automate through typed contracts, least privilege, deterministic gates, canaries, bounded authority, and reversible changes.

Implementation sketch

```
attestation_policy:
  workload_digest: sha256:...
  signer: approved-release-identity
  platform:
    debug_disabled: true
    security_version_min: 7
  freshness:
    nonce_required: true
    max_age_seconds: 60
  decision:
    on_pass: release-model-decryption-key
    on_fail: deny-and-record
  evidence:
    retain: [eat-token, verifier-result, endorsements, reference-values]
  limitations:
    - availability-not-guaranteed
    - approved-code-can-still-leak-output
```

Failure patterns

Treating tee architecture and isolation as a product feature instead of an end-to-end engineering system.

Relying on intended policy, rule code, or controller state without proving applied and operational behavior.

Ignoring data quality, identity, time, cryptographic lifecycle, supplier state, 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 result without negative tests, rollback, and recurrence monitoring.

Validation and evidence

Method	Expected evidence
Examine	Requirements, architecture, identities, data, algorithms, policy, configuration, ownership, and dependencies for tee architecture and isolation.
Observe	Active state, applied policy, raw events, telemetry, artifacts, cryptographic results, resources, and endpoint behavior.
Test	Expected behavior, prohibited behavior, malformed inputs, dependency loss, partial failure, scale, recovery, and rollback.
Measure	Availability, latency, detection quality, evidence completeness, cryptographic cost, risk, resource use, and mission outcome.
Reconcile	Intended state against observed security, identity, application, cloud, cryptographic, data, and evidence records.

1.3 Threat model and trusted computing base

Threat model and trusted computing base must be implemented as an observable engineering mechanism, not a product label or maturity claim. Identify host, hypervisor, cloud operator, firmware, hardware, side channel, denial of service, workload, compiler, and supply-chain assumptions. The design should name authoritative inputs, identities, state transitions, dependencies, cryptographic or policy boundaries, limits, owners, telemetry, failure behavior, and acceptance criteria.

This guide traces the subject through the workload, TEE, evidence, verifier, appraisal, key release, protected computation, output, and relying-party decision chain. A configured integration, successful API response, established cryptographic session, green dashboard, or closed ticket is not sufficient proof. Intended state must be reconciled with applied state, negative tests, degraded behavior, raw evidence, recovery, and the resulting security or mission 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 authorized and prohibited behavior, inject realistic failures, measure convergence or recovery, and retain a tested rollback. Automation must stop safely when authority, freshness, identity, integrity, cryptographic state, or observed outcomes are ambiguous.

Troubleshooting locates the first divergence from the expected state machine or end-to-end transaction. Inspect every authority boundary and account for time, caching, eventual consistency, stale identity, incomplete collection, parser drift, asymmetric

paths, encapsulation, supply-chain state, side channels, partial failure, and missing evidence. Closure requires causal explanation, reproducibility, validated restoration, residual limitations, and prevention or monitoring actions.

Engineering practices

- Define authority, ownership, scope, dependencies, limits, and acceptance criteria for threat model and trusted computing base.
- Model expected state, trust boundaries, data flow, identity, and failure behavior before implementation.
- Validate intended configuration, active state, applied enforcement, raw evidence, and end-to-end outcome independently.
- Test normal, prohibited, degraded, failed, scale, recovery, and rollback conditions.
- Preserve policy, source, version, identity, telemetry, artifact, packet, log, time, and decision evidence.
- Automate through typed contracts, least privilege, deterministic gates, canaries, bounded authority, and reversible changes.

Failure patterns

- Treating threat model and trusted computing base as a product feature instead of an end-to-end engineering system.
- Relying on intended policy, rule code, or controller state without proving applied and operational behavior.
- Ignoring data quality, identity, time, cryptographic lifecycle, supplier state, 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 result without negative tests, rollback, and recurrence monitoring.

Validation and evidence

Method	Expected evidence
Examine	Requirements, architecture, identities, data, algorithms, policy, configuration, ownership, and dependencies for threat model and trusted computing base.
Observe	Active state, applied policy, raw events, telemetry, artifacts, cryptographic results, resources, and endpoint behavior.
Test	Expected behavior, prohibited behavior, malformed inputs, dependency loss, partial failure, scale, recovery, and rollback.
Measure	Availability, latency, detection quality, evidence completeness, cryptographic cost, risk, resource use, and mission outcome.
Reconcile	Intended state against observed security, identity, application, cloud, cryptographic, data, and evidence records.

1.4 Platform diversity and portability

Platform diversity and portability must be implemented as an observable engineering mechanism, not a product label or maturity claim. Compare enclave, confidential VM, realm, secure processor, GPU, accelerator, embedded, and cloud service models. The design should name authoritative inputs, identities, state transitions, dependencies, cryptographic or policy boundaries, limits, owners, telemetry, failure behavior, and acceptance criteria.

This guide traces the subject through the workload, TEE, evidence, verifier, appraisal, key release, protected computation, output, and relying-party decision chain. A configured integration, successful API response, established cryptographic session, green dashboard, or closed ticket is not sufficient proof. Intended state must be reconciled with applied state, negative tests, degraded behavior, raw evidence, recovery, and the resulting security or mission 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 authorized and prohibited behavior, inject realistic failures, measure convergence or recovery, and retain a tested rollback. Automation must stop safely when authority, freshness, identity, integrity, cryptographic state, or observed outcomes are ambiguous.

Troubleshooting locates the first divergence from the expected state machine or end-to-end transaction. Inspect every authority boundary and account for time, caching, eventual consistency, stale identity, incomplete collection, parser drift, asymmetric paths, encapsulation, supply-chain state, side channels, partial failure, and missing evidence. Closure requires causal explanation, reproducibility, validated restoration, residual limitations, and prevention or monitoring actions.

Engineering practices

Define authority, ownership, scope, dependencies, limits, and acceptance criteria for platform diversity and portability.

Model expected state, trust boundaries, data flow, identity, and failure behavior before implementation.

Validate intended configuration, active state, applied enforcement, raw evidence, and end-to-end outcome independently.

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

Preserve policy, source, version, identity, telemetry, artifact, packet, log, time, and decision evidence.

Automate through typed contracts, least privilege, deterministic gates, canaries, bounded authority, and reversible changes.

Failure patterns

Treating platform diversity and portability as a product feature instead of an end-to-end engineering system.

Relying on intended policy, rule code, or controller state without proving applied and operational behavior.

Ignoring data quality, identity, time, cryptographic lifecycle, supplier state, 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 result without negative tests, rollback, and recurrence monitoring.

Validation and evidence

Method	Expected evidence
Examine	Requirements, architecture, identities, data, algorithms, policy, configuration, ownership, and dependencies for platform diversity and portability.
Observe	Active state, applied policy, raw events, telemetry, artifacts, cryptographic results, resources, and endpoint behavior.
Test	Expected behavior, prohibited behavior, malformed inputs, dependency loss, partial failure, scale, recovery, and rollback.
Measure	Availability, latency, detection quality, evidence completeness, cryptographic cost, risk, resource use, and mission outcome.
Reconcile	Intended state against observed security, identity, application, cloud, cryptographic, data, and evidence records.

Chapter 01 laboratory and review

Practical laboratory

Create a threat model for a confidential workload and list exactly which actors, data, code, metadata, I/O, availability, and side channels are protected or exposed.

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 confidential computing to agentic AI, edge devices, robotics, GPUs, DPUs, and multi-party sovereign infrastructure.

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

Remote attestation architecture and evidence

Establish how a relying party can evaluate the state and identity of a protected environment.

Chapter operating position
Establish how a relying party can evaluate the state and identity of a protected environment.

2.1 Attester, verifier, and relying party

Attester, verifier, and relying party must be implemented as an observable engineering mechanism, not a product label or maturity claim. Use RATS roles, evidence, endorsements, reference values, appraisal policies, attestation results, and trust decisions. The design should name authoritative inputs, identities, state transitions, dependencies, cryptographic or policy boundaries, limits, owners, telemetry, failure behavior, and acceptance criteria.

This guide traces the subject through the workload, TEE, evidence, verifier, appraisal, key release, protected computation, output, and relying-party decision chain. A configured integration, successful API response, established cryptographic session, green dashboard, or closed ticket is not sufficient proof. Intended state must be reconciled with applied state, negative tests, degraded behavior, raw evidence, recovery, and the resulting security or mission 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 authorized and prohibited behavior, inject realistic failures, measure convergence or recovery, and retain a tested rollback. Automation must stop safely when authority, freshness, identity, integrity, cryptographic state, or observed outcomes are ambiguous.

Troubleshooting locates the first divergence from the expected state machine or end-to-end transaction. Inspect every authority boundary and account for time, caching, eventual consistency, stale identity, incomplete collection, parser drift, asymmetric paths, encapsulation, supply-chain state, side channels, partial failure, and missing evidence. Closure requires causal explanation, reproducibility, validated restoration, residual limitations, and prevention or monitoring actions.

Engineering practices

- Define authority, ownership, scope, dependencies, limits, and acceptance criteria for attester, verifier, and relying party.
- Model expected state, trust boundaries, data flow, identity, and failure behavior before implementation.
- Validate intended configuration, active state, applied enforcement, raw evidence, and end-to-end outcome independently.
- Test normal, prohibited, degraded, failed, scale, recovery, and rollback conditions.
- Preserve policy, source, version, identity, telemetry, artifact, packet, log, time, and decision evidence.
- Automate through typed contracts, least privilege, deterministic gates, canaries, bounded authority, and reversible changes.

Failure patterns

- Treating attester, verifier, and relying party as a product feature instead of an end-to-end engineering system.
- Relying on intended policy, rule code, or controller state without proving applied and operational behavior.
- Ignoring data quality, identity, time, cryptographic lifecycle, supplier state, 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 result without negative tests, rollback, and recurrence monitoring.

Validation and evidence

Method	Expected evidence
Examine	Requirements, architecture, identities, data, algorithms, policy, configuration, ownership, and dependencies for attester, verifier, and relying party.
Observe	Active state, applied policy, raw events, telemetry, artifacts, cryptographic results, resources, and endpoint behavior.
Test	Expected behavior, prohibited behavior, malformed inputs, dependency loss, partial failure, scale, recovery, and rollback.
Measure	Availability, latency, detection quality, evidence completeness, cryptographic cost, risk, resource use, and mission outcome.
Reconcile	Intended state against observed security, identity, application, cloud, cryptographic, data, and evidence records.

2.2 Measurements and claims

Measurements and claims must be implemented as an observable engineering mechanism, not a product label or maturity claim. Represent hardware, firmware, configuration, boot, workload, debug, lifecycle, identity, freshness, and platform properties. The design should name authoritative inputs, identities, state transitions, dependencies, cryptographic or policy boundaries, limits, owners, telemetry, failure behavior, and acceptance criteria.

This guide traces the subject through the workload, TEE, evidence, verifier, appraisal, key release, protected computation, output, and relying-party decision chain. A configured integration, successful API response, established cryptographic session, green dashboard, or closed ticket is not sufficient proof. Intended state must be reconciled with applied state, negative tests, degraded behavior, raw evidence, recovery, and the resulting security or mission 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 authorized and prohibited behavior, inject realistic failures, measure convergence or recovery, and retain a tested rollback. Automation must stop safely when authority, freshness, identity, integrity, cryptographic state, or observed outcomes are ambiguous.

Troubleshooting locates the first divergence from the expected state machine or end-to-end transaction. Inspect every authority boundary and account for time, caching, eventual consistency, stale identity, incomplete collection, parser drift, asymmetric paths, encapsulation, supply-chain state, side channels, partial failure, and missing evidence. Closure requires causal explanation, reproducibility, validated restoration, residual limitations, and prevention or monitoring actions.

Engineering practices

- Define authority, ownership, scope, dependencies, limits, and acceptance criteria for measurements and claims.
- Model expected state, trust boundaries, data flow, identity, and failure behavior before implementation.
- Validate intended configuration, active state, applied enforcement, raw evidence, and end-to-end outcome independently.
- Test normal, prohibited, degraded, failed, scale, recovery, and rollback conditions.
- Preserve policy, source, version, identity, telemetry, artifact, packet, log, time, and decision evidence.
- Automate through typed contracts, least privilege, deterministic gates, canaries, bounded authority, and reversible changes.

Implementation sketch

```
attestation_policy:
  workload_digest: sha256:...
  signer: approved-release-identity
  platform:
    debug_disabled: true
    security_version_min: 7
  freshness:
    nonce_required: true
    max_age_seconds: 60
  decision:
    on_pass: release-model-decryption-key
    on_fail: deny-and-record
  evidence:
    retain: [eat-token, verifier-result, endorsements, reference-values]
  limitations:
    - availability-not-guaranteed
    - approved-code-can-still-leak-output
```

Failure patterns

Treating measurements and claims as a product feature instead of an end-to-end engineering system.

Relying on intended policy, rule code, or controller state without proving applied and operational behavior.

Ignoring data quality, identity, time, cryptographic lifecycle, supplier state, 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 result without negative tests, rollback, and recurrence monitoring.

Validation and evidence

Method	Expected evidence
Examine	Requirements, architecture, identities, data, algorithms, policy, configuration, ownership, and dependencies for measurements and claims.
Observe	Active state, applied policy, raw events, telemetry, artifacts, cryptographic results, resources, and endpoint behavior.
Test	Expected behavior, prohibited behavior, malformed inputs, dependency loss, partial failure, scale, recovery, and rollback.
Measure	Availability, latency, detection quality, evidence completeness, cryptographic cost, risk, resource use, and mission outcome.
Reconcile	Intended state against observed security, identity, application, cloud, cryptographic, data, and evidence records.

2.3 Entity Attestation Token

Entity Attestation Token must be implemented as an observable engineering mechanism, not a product label or maturity claim. Use EAT claims in CWT or JWT form, nonces, identifiers, submodules, detached data, signing, and media types. The design should name authoritative inputs, identities, state transitions, dependencies, cryptographic or policy boundaries, limits, owners, telemetry, failure behavior, and acceptance criteria.

This guide traces the subject through the workload, TEE, evidence, verifier, appraisal, key release, protected computation, output, and relying-party decision chain. A configured integration, successful API response, established cryptographic session, green dashboard, or closed ticket is not sufficient proof. Intended state must be reconciled with applied state, negative tests, degraded behavior, raw evidence, recovery, and the resulting security or mission 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 authorized and prohibited behavior, inject realistic failures, measure convergence or recovery, and retain a tested rollback. Automation must stop safely when authority, freshness, identity, integrity, cryptographic state, or observed outcomes are ambiguous.

Troubleshooting locates the first divergence from the expected state machine or end-to-end transaction. Inspect every authority boundary and account for time, caching, eventual consistency, stale identity, incomplete collection, parser drift, asymmetric

paths, encapsulation, supply-chain state, side channels, partial failure, and missing evidence. Closure requires causal explanation, reproducibility, validated restoration, residual limitations, and prevention or monitoring actions.

Engineering practices

- Define authority, ownership, scope, dependencies, limits, and acceptance criteria for entity attestation token.
- Model expected state, trust boundaries, data flow, identity, and failure behavior before implementation.
- Validate intended configuration, active state, applied enforcement, raw evidence, and end-to-end outcome independently.
- Test normal, prohibited, degraded, failed, scale, recovery, and rollback conditions.
- Preserve policy, source, version, identity, telemetry, artifact, packet, log, time, and decision evidence.
- Automate through typed contracts, least privilege, deterministic gates, canaries, bounded authority, and reversible changes.

Failure patterns

- Treating entity attestation token as a product feature instead of an end-to-end engineering system.
- Relying on intended policy, rule code, or controller state without proving applied and operational behavior.
- Ignoring data quality, identity, time, cryptographic lifecycle, supplier state, 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 result without negative tests, rollback, and recurrence monitoring.

Validation and evidence

Method	Expected evidence
Examine	Requirements, architecture, identities, data, algorithms, policy, configuration, ownership, and dependencies for entity attestation token.
Observe	Active state, applied policy, raw events, telemetry, artifacts, cryptographic results, resources, and endpoint behavior.
Test	Expected behavior, prohibited behavior, malformed inputs, dependency loss, partial failure, scale, recovery, and rollback.
Measure	Availability, latency, detection quality, evidence completeness, cryptographic cost, risk, resource use, and mission outcome.
Reconcile	Intended state against observed security, identity, application, cloud, cryptographic, data, and evidence records.

2.4 Replay, freshness, privacy, and trust

Replay, freshness, privacy, and trust must be implemented as an observable engineering mechanism, not a product label or maturity claim. Use challenge, time, epoch, keys, endorsements, reference updates, pseudonymity, and minimal claims. The design should name authoritative inputs, identities, state transitions, dependencies, cryptographic or policy boundaries, limits, owners, telemetry, failure behavior, and acceptance criteria.

This guide traces the subject through the workload, TEE, evidence, verifier, appraisal, key release, protected computation, output, and relying-party decision chain. A configured integration, successful API response, established cryptographic session, green dashboard, or closed ticket is not sufficient proof. Intended state must be reconciled with applied state, negative tests, degraded behavior, raw evidence, recovery, and the resulting security or mission 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 authorized and prohibited behavior, inject realistic failures, measure convergence or recovery, and retain a tested rollback. Automation must stop safely when authority, freshness, identity, integrity, cryptographic state, or observed outcomes are ambiguous.

Troubleshooting locates the first divergence from the expected state machine or end-to-end transaction. Inspect every authority boundary and account for time, caching, eventual consistency, stale identity, incomplete collection, parser drift, asymmetric paths, encapsulation, supply-chain state, side channels, partial failure, and missing evidence. Closure requires causal explanation, reproducibility, validated restoration, residual limitations, and prevention or monitoring actions.

Engineering practices

Define authority, ownership, scope, dependencies, limits, and acceptance criteria for replay, freshness, privacy, and trust.

Model expected state, trust boundaries, data flow, identity, and failure behavior before implementation.

Validate intended configuration, active state, applied enforcement, raw evidence, and end-to-end outcome independently.

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

Preserve policy, source, version, identity, telemetry, artifact, packet, log, time, and decision evidence.

Automate through typed contracts, least privilege, deterministic gates, canaries, bounded authority, and reversible changes.

Failure patterns

Treating replay, freshness, privacy, and trust as a product feature instead of an end-to-end engineering system.

Relying on intended policy, rule code, or controller state without proving applied and operational behavior.

Ignoring data quality, identity, time, cryptographic lifecycle, supplier state, 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 result without negative tests, rollback, and recurrence monitoring.

Validation and evidence

Method	Expected evidence
Examine	Requirements, architecture, identities, data, algorithms, policy, configuration, ownership, and dependencies for replay, freshness, privacy, and trust.
Observe	Active state, applied policy, raw events, telemetry, artifacts, cryptographic results, resources, and endpoint behavior.
Test	Expected behavior, prohibited behavior, malformed inputs, dependency loss, partial failure, scale, recovery, and rollback.
Measure	Availability, latency, detection quality, evidence completeness, cryptographic cost, risk, resource use, and mission outcome.
Reconcile	Intended state against observed security, identity, application, cloud, cryptographic, data, and evidence records.

Chapter 02 laboratory and review

Practical laboratory

Model and simulate an attestation exchange from challenge through evidence, verification, appraisal, result, relying-party decision, and denial.

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 layered and composite attestation, attested network devices, confidential AI evidence, and post-quantum attestation signatures.

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

Attested workload lifecycle and key release

Bind sensitive data and authority to verified code, configuration, and execution state.

Chapter operating position
Bind sensitive data and authority to verified code, configuration, and execution state.

3.1 Image, measurement, and policy

Image, measurement, and policy must be implemented as an observable engineering mechanism, not a product label or maturity claim. Define approved workload identity, digest, signer, configuration, runtime, platform, version, debug state, and policy. The design should name authoritative inputs, identities, state transitions, dependencies, cryptographic or policy boundaries, limits, owners, telemetry, failure behavior, and acceptance criteria.

This guide traces the subject through the workload, TEE, evidence, verifier, appraisal, key release, protected computation, output, and relying-party decision chain. A configured integration, successful API response, established cryptographic session, green dashboard, or closed ticket is not sufficient proof. Intended state must be reconciled with applied state, negative tests, degraded behavior, raw evidence, recovery, and the resulting security or mission 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 authorized and prohibited behavior, inject realistic failures, measure convergence or recovery, and retain a tested rollback. Automation must stop safely when authority, freshness, identity, integrity, cryptographic state, or observed outcomes are ambiguous.

Troubleshooting locates the first divergence from the expected state machine or end-to-end transaction. Inspect every authority boundary and account for time, caching, eventual consistency, stale identity, incomplete collection, parser drift, asymmetric paths, encapsulation, supply-chain state, side channels, partial failure, and missing evidence. Closure requires causal explanation, reproducibility, validated restoration, residual limitations, and prevention or monitoring actions.

Engineering practices

- Define authority, ownership, scope, dependencies, limits, and acceptance criteria for image, measurement, and policy.
- Model expected state, trust boundaries, data flow, identity, and failure behavior before implementation.
- Validate intended configuration, active state, applied enforcement, raw evidence, and end-to-end outcome independently.
- Test normal, prohibited, degraded, failed, scale, recovery, and rollback conditions.
- Preserve policy, source, version, identity, telemetry, artifact, packet, log, time, and decision evidence.
- Automate through typed contracts, least privilege, deterministic gates, canaries, bounded authority, and reversible changes.

Failure patterns

- Treating image, measurement, and policy as a product feature instead of an end-to-end engineering system.
- Relying on intended policy, rule code, or controller state without proving applied and operational behavior.
- Ignoring data quality, identity, time, cryptographic lifecycle, supplier state, 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 result without negative tests, rollback, and recurrence monitoring.

Validation and evidence

Method	Expected evidence
Examine	Requirements, architecture, identities, data, algorithms, policy, configuration, ownership, and dependencies for image, measurement, and policy.
Observe	Active state, applied policy, raw events, telemetry, artifacts, cryptographic results, resources, and endpoint behavior.
Test	Expected behavior, prohibited behavior, malformed inputs, dependency loss, partial failure, scale, recovery, and rollback.
Measure	Availability, latency, detection quality, evidence completeness, cryptographic cost, risk, resource use, and mission outcome.
Reconcile	Intended state against observed security, identity, application, cloud, cryptographic, data, and evidence records.

3.2 Secret and data release

Secret and data release must be implemented as an observable engineering mechanism, not a product label or maturity claim. Release keys, credentials, model weights, datasets, licenses, or tokens only after successful attestation and authorization. The design should name authoritative inputs, identities, state transitions, dependencies, cryptographic or policy boundaries, limits, owners, telemetry, failure behavior, and acceptance criteria.

This guide traces the subject through the workload, TEE, evidence, verifier, appraisal, key release, protected computation, output, and relying-party decision chain. A configured integration, successful API response, established cryptographic session, green dashboard, or closed ticket is not sufficient proof. Intended state must be reconciled with applied state, negative tests, degraded behavior, raw evidence, recovery, and the resulting security or mission 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 authorized and prohibited behavior, inject realistic failures, measure convergence or recovery, and retain a tested rollback. Automation must stop safely when authority, freshness, identity, integrity, cryptographic state, or observed outcomes are ambiguous.

Troubleshooting locates the first divergence from the expected state machine or end-to-end transaction. Inspect every authority boundary and account for time, caching, eventual consistency, stale identity, incomplete collection, parser drift, asymmetric paths, encapsulation, supply-chain state, side channels, partial failure, and missing evidence. Closure requires causal explanation, reproducibility, validated restoration, residual limitations, and prevention or monitoring actions.

Engineering practices

- Define authority, ownership, scope, dependencies, limits, and acceptance criteria for secret and data release.
- Model expected state, trust boundaries, data flow, identity, and failure behavior before implementation.
- Validate intended configuration, active state, applied enforcement, raw evidence, and end-to-end outcome independently.
- Test normal, prohibited, degraded, failed, scale, recovery, and rollback conditions.
- Preserve policy, source, version, identity, telemetry, artifact, packet, log, time, and decision evidence.
- Automate through typed contracts, least privilege, deterministic gates, canaries, bounded authority, and reversible changes.

Implementation sketch

```
attestation_policy:
  workload_digest: sha256:...
  signer: approved-release-identity
  platform:
    debug_disabled: true
    security_version_min: 7
  freshness:
    nonce_required: true
    max_age_seconds: 60
  decision:
    on_pass: release-model-decryption-key
    on_fail: deny-and-record
  evidence:
    retain: [eat-token, verifier-result, endorsements, reference-values]
  limitations:
    - availability-not-guaranteed
    - approved-code-can-still-leak-output
```

Failure patterns

Treating secret and data release as a product feature instead of an end-to-end engineering system.

Relying on intended policy, rule code, or controller state without proving applied and operational behavior.

Ignoring data quality, identity, time, cryptographic lifecycle, supplier state, 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 result without negative tests, rollback, and recurrence monitoring.

Validation and evidence

Method	Expected evidence
Examine	Requirements, architecture, identities, data, algorithms, policy, configuration, ownership, and dependencies for secret and data release.
Observe	Active state, applied policy, raw events, telemetry, artifacts, cryptographic results, resources, and endpoint behavior.
Test	Expected behavior, prohibited behavior, malformed inputs, dependency loss, partial failure, scale, recovery, and rollback.
Measure	Availability, latency, detection quality, evidence completeness, cryptographic cost, risk, resource use, and mission outcome.
Reconcile	Intended state against observed security, identity, application, cloud, cryptographic, data, and evidence records.

3.3 Renewal, update, and revocation

Renewal, update, and revocation must be implemented as an observable engineering mechanism, not a product label or maturity claim. Handle patching, measurement change, rolling deployment, grace periods, reference values, compromised versions, and rollback. The design should name authoritative inputs, identities, state transitions, dependencies, cryptographic or policy boundaries, limits, owners, telemetry, failure behavior, and acceptance criteria.

This guide traces the subject through the workload, TEE, evidence, verifier, appraisal, key release, protected computation, output, and relying-party decision chain. A configured integration, successful API response, established cryptographic session, green dashboard, or closed ticket is not sufficient proof. Intended state must be reconciled with applied state, negative tests, degraded behavior, raw evidence, recovery, and the resulting security or mission 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 authorized and prohibited behavior, inject realistic failures, measure convergence or recovery, and retain a tested rollback. Automation must stop safely when authority, freshness, identity, integrity, cryptographic state, or observed outcomes are ambiguous.

Troubleshooting locates the first divergence from the expected state machine or end-to-end transaction. Inspect every authority boundary and account for time, caching, eventual consistency, stale identity, incomplete collection, parser drift, asymmetric

paths, encapsulation, supply-chain state, side channels, partial failure, and missing evidence. Closure requires causal explanation, reproducibility, validated restoration, residual limitations, and prevention or monitoring actions.

Engineering practices

- Define authority, ownership, scope, dependencies, limits, and acceptance criteria for renewal, update, and revocation.
- Model expected state, trust boundaries, data flow, identity, and failure behavior before implementation.
- Validate intended configuration, active state, applied enforcement, raw evidence, and end-to-end outcome independently.
- Test normal, prohibited, degraded, failed, scale, recovery, and rollback conditions.
- Preserve policy, source, version, identity, telemetry, artifact, packet, log, time, and decision evidence.
- Automate through typed contracts, least privilege, deterministic gates, canaries, bounded authority, and reversible changes.

Failure patterns

- Treating renewal, update, and revocation as a product feature instead of an end-to-end engineering system.
- Relying on intended policy, rule code, or controller state without proving applied and operational behavior.
- Ignoring data quality, identity, time, cryptographic lifecycle, supplier state, 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 result without negative tests, rollback, and recurrence monitoring.

Validation and evidence

Method	Expected evidence
Examine	Requirements, architecture, identities, data, algorithms, policy, configuration, ownership, and dependencies for renewal, update, and revocation.
Observe	Active state, applied policy, raw events, telemetry, artifacts, cryptographic results, resources, and endpoint behavior.
Test	Expected behavior, prohibited behavior, malformed inputs, dependency loss, partial failure, scale, recovery, and rollback.
Measure	Availability, latency, detection quality, evidence completeness, cryptographic cost, risk, resource use, and mission outcome.
Reconcile	Intended state against observed security, identity, application, cloud, cryptographic, data, and evidence records.

3.4 Failure and recovery

Failure and recovery must be implemented as an observable engineering mechanism, not a product label or maturity claim. Design verifier outage, endorsement loss, expired evidence, platform replacement, state loss, backup, and emergency access. The design should name authoritative inputs, identities, state transitions, dependencies, cryptographic or policy boundaries, limits, owners, telemetry, failure behavior, and acceptance criteria.

This guide traces the subject through the workload, TEE, evidence, verifier, appraisal, key release, protected computation, output, and relying-party decision chain. A configured integration, successful API response, established cryptographic session, green dashboard, or closed ticket is not sufficient proof. Intended state must be reconciled with applied state, negative tests, degraded behavior, raw evidence, recovery, and the resulting security or mission 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 authorized and prohibited behavior, inject realistic failures, measure convergence or recovery, and retain a tested rollback. Automation must stop safely when authority, freshness, identity, integrity, cryptographic state, or observed outcomes are ambiguous.

Troubleshooting locates the first divergence from the expected state machine or end-to-end transaction. Inspect every authority boundary and account for time, caching, eventual consistency, stale identity, incomplete collection, parser drift, asymmetric paths, encapsulation, supply-chain state, side channels, partial failure, and missing evidence. Closure requires causal explanation, reproducibility, validated restoration, residual limitations, and prevention or monitoring actions.

Engineering practices

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

Model expected state, trust boundaries, data flow, identity, and failure behavior before implementation.

Validate intended configuration, active state, applied enforcement, raw evidence, and end-to-end outcome independently.

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

Preserve policy, source, version, identity, telemetry, artifact, packet, log, time, and decision evidence.

Automate through typed contracts, least privilege, deterministic gates, canaries, bounded authority, and reversible changes.

Failure patterns

Treating failure and recovery as a product feature instead of an end-to-end engineering system.

Relying on intended policy, rule code, or controller state without proving applied and operational behavior.

Ignoring data quality, identity, time, cryptographic lifecycle, supplier state, 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 result without negative tests, rollback, and recurrence monitoring.

Validation and evidence

Method	Expected evidence
Examine	Requirements, architecture, identities, data, algorithms, policy, configuration, ownership, and dependencies for failure and recovery.
Observe	Active state, applied policy, raw events, telemetry, artifacts, cryptographic results, resources, and endpoint behavior.
Test	Expected behavior, prohibited behavior, malformed inputs, dependency loss, partial failure, scale, recovery, and rollback.
Measure	Availability, latency, detection quality, evidence completeness, cryptographic cost, risk, resource use, and mission outcome.
Reconcile	Intended state against observed security, identity, application, cloud, cryptographic, data, and evidence records.

Chapter 03 laboratory and review

Practical laboratory
Build an attestation-gated key-release workflow and test approved, outdated, debug-enabled, replayed, revoked, and unavailable-verifier cases.

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 policy-bound data capsules, threshold key release, verifiable updates, and autonomous confidential workloads.

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

Confidential cloud, containers, AI, and accelerators

Integrate TEEs into real platform, workload, and data pipelines.

Chapter operating position
Integrate TEEs into real platform, workload, and data pipelines.

4.1 Confidential VMs and containers

Confidential VMs and containers must be implemented as an observable engineering mechanism, not a product label or maturity claim. Protect images, boot, memory, workload identity, volumes, networks, orchestration, attestation, and node lifecycle. The design should name authoritative inputs, identities, state transitions, dependencies, cryptographic or policy boundaries, limits, owners, telemetry, failure behavior, and acceptance criteria.

This guide traces the subject through the workload, TEE, evidence, verifier, appraisal, key release, protected computation, output, and relying-party decision chain. A configured integration, successful API response, established cryptographic session, green dashboard, or closed ticket is not sufficient proof. Intended state must be reconciled with applied state, negative tests, degraded behavior, raw evidence, recovery, and the resulting security or mission 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 authorized and prohibited behavior, inject realistic failures, measure convergence or recovery, and retain a tested rollback. Automation must stop safely when authority, freshness, identity, integrity, cryptographic state, or observed outcomes are ambiguous.

Troubleshooting locates the first divergence from the expected state machine or end-to-end transaction. Inspect every authority boundary and account for time, caching, eventual consistency, stale identity, incomplete collection, parser drift, asymmetric paths, encapsulation, supply-chain state, side channels, partial failure, and missing evidence. Closure requires causal explanation, reproducibility, validated restoration, residual limitations, and prevention or monitoring actions.

Engineering practices

- Define authority, ownership, scope, dependencies, limits, and acceptance criteria for confidential vms and containers.
- Model expected state, trust boundaries, data flow, identity, and failure behavior before implementation.
- Validate intended configuration, active state, applied enforcement, raw evidence, and end-to-end outcome independently.
- Test normal, prohibited, degraded, failed, scale, recovery, and rollback conditions.
- Preserve policy, source, version, identity, telemetry, artifact, packet, log, time, and decision evidence.
- Automate through typed contracts, least privilege, deterministic gates, canaries, bounded authority, and reversible changes.

Failure patterns

- Treating confidential vms and containers as a product feature instead of an end-to-end engineering system.
- Relying on intended policy, rule code, or controller state without proving applied and operational behavior.
- Ignoring data quality, identity, time, cryptographic lifecycle, supplier state, 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 result without negative tests, rollback, and recurrence monitoring.

Validation and evidence

Method	Expected evidence
Examine	Requirements, architecture, identities, data, algorithms, policy, configuration, ownership, and dependencies for confidential vms and containers.
Observe	Active state, applied policy, raw events, telemetry, artifacts, cryptographic results, resources, and endpoint behavior.
Test	Expected behavior, prohibited behavior, malformed inputs, dependency loss, partial failure, scale, recovery, and rollback.
Measure	Availability, latency, detection quality, evidence completeness, cryptographic cost, risk, resource use, and mission outcome.
Reconcile	Intended state against observed security, identity, application, cloud, cryptographic, data, and evidence records.

4.2 Confidential AI training and inference

Confidential AI training and inference must be implemented as an observable engineering mechanism, not a product label or maturity claim. Protect data, model weights, prompts, intermediate state, outputs, operators, GPU memory, distributed jobs, and telemetry. The design should name authoritative inputs, identities, state transitions, dependencies, cryptographic or policy boundaries, limits, owners, telemetry, failure behavior, and acceptance criteria.

This guide traces the subject through the workload, TEE, evidence, verifier, appraisal, key release, protected computation, output, and relying-party decision chain. A configured integration, successful API response, established cryptographic session, green dashboard, or closed ticket is not sufficient proof. Intended state must be reconciled with applied state, negative tests, degraded behavior, raw evidence, recovery, and the resulting security or mission 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 authorized and prohibited behavior, inject realistic failures, measure convergence or recovery, and retain a tested rollback. Automation must stop safely when authority, freshness, identity, integrity, cryptographic state, or observed outcomes are ambiguous.

Troubleshooting locates the first divergence from the expected state machine or end-to-end transaction. Inspect every authority boundary and account for time, caching, eventual consistency, stale identity, incomplete collection, parser drift, asymmetric paths, encapsulation, supply-chain state, side channels, partial failure, and missing evidence. Closure requires causal explanation, reproducibility, validated restoration, residual limitations, and prevention or monitoring actions.

Engineering practices

- Define authority, ownership, scope, dependencies, limits, and acceptance criteria for confidential ai training and inference.
- Model expected state, trust boundaries, data flow, identity, and failure behavior before implementation.
- Validate intended configuration, active state, applied enforcement, raw evidence, and end-to-end outcome independently.
- Test normal, prohibited, degraded, failed, scale, recovery, and rollback conditions.
- Preserve policy, source, version, identity, telemetry, artifact, packet, log, time, and decision evidence.
- Automate through typed contracts, least privilege, deterministic gates, canaries, bounded authority, and reversible changes.

Implementation sketch

```
attestation_policy:
  workload_digest: sha256:...
  signer: approved-release-identity
  platform:
    debug_disabled: true
    security_version_min: 7
  freshness:
    nonce_required: true
    max_age_seconds: 60
  decision:
    on_pass: release-model-decryption-key
    on_fail: deny-and-record
  evidence:
    retain: [eat-token, verifier-result, endorsements, reference-values]
  limitations:
    - availability-not-guaranteed
    - approved-code-can-still-leak-output
```

Failure patterns

- Treating confidential ai training and inference as a product feature instead of an end-to-end engineering system.
- Relying on intended policy, rule code, or controller state without proving applied and operational behavior.
- Ignoring data quality, identity, time, cryptographic lifecycle, supplier state, 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 result without negative tests, rollback, and recurrence monitoring.

Validation and evidence

Method	Expected evidence
Examine	Requirements, architecture, identities, data, algorithms, policy, configuration, ownership, and dependencies for confidential ai training and inference.
Observe	Active state, applied policy, raw events, telemetry, artifacts, cryptographic results, resources, and endpoint behavior.
Test	Expected behavior, prohibited behavior, malformed inputs, dependency loss, partial failure, scale, recovery, and rollback.
Measure	Availability, latency, detection quality, evidence completeness, cryptographic cost, risk, resource use, and mission outcome.
Reconcile	Intended state against observed security, identity, application, cloud, cryptographic, data, and evidence records.

4.3 I/O and device trust

I/O and device trust must be implemented as an observable engineering mechanism, not a product label or maturity claim. Assess storage, network, GPUs, accelerators, passthrough, DMA, trusted paths, encryption, and device attestation. The design should name authoritative inputs, identities, state transitions, dependencies, cryptographic or policy boundaries, limits, owners, telemetry, failure behavior, and acceptance criteria.

This guide traces the subject through the workload, TEE, evidence, verifier, appraisal, key release, protected computation, output, and relying-party decision chain. A configured integration, successful API response, established cryptographic session, green dashboard, or closed ticket is not sufficient proof. Intended state must be reconciled with applied state, negative tests, degraded behavior, raw evidence, recovery, and the resulting security or mission 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 authorized and prohibited behavior, inject realistic failures, measure convergence or recovery, and retain a tested rollback. Automation must stop safely when authority, freshness, identity, integrity, cryptographic state, or observed outcomes are ambiguous.

Troubleshooting locates the first divergence from the expected state machine or end-to-end transaction. Inspect every authority boundary and account for time, caching, eventual consistency, stale identity, incomplete collection, parser drift, asymmetric

paths, encapsulation, supply-chain state, side channels, partial failure, and missing evidence. Closure requires causal explanation, reproducibility, validated restoration, residual limitations, and prevention or monitoring actions.

Engineering practices

- Define authority, ownership, scope, dependencies, limits, and acceptance criteria for i/o and device trust.
- Model expected state, trust boundaries, data flow, identity, and failure behavior before implementation.
- Validate intended configuration, active state, applied enforcement, raw evidence, and end-to-end outcome independently.
- Test normal, prohibited, degraded, failed, scale, recovery, and rollback conditions.
- Preserve policy, source, version, identity, telemetry, artifact, packet, log, time, and decision evidence.
- Automate through typed contracts, least privilege, deterministic gates, canaries, bounded authority, and reversible changes.

Failure patterns

- Treating i/o and device trust as a product feature instead of an end-to-end engineering system.
- Relying on intended policy, rule code, or controller state without proving applied and operational behavior.
- Ignoring data quality, identity, time, cryptographic lifecycle, supplier state, 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 result without negative tests, rollback, and recurrence monitoring.

Validation and evidence

Method	Expected evidence
Examine	Requirements, architecture, identities, data, algorithms, policy, configuration, ownership, and dependencies for i/o and device trust.
Observe	Active state, applied policy, raw events, telemetry, artifacts, cryptographic results, resources, and endpoint behavior.
Test	Expected behavior, prohibited behavior, malformed inputs, dependency loss, partial failure, scale, recovery, and rollback.
Measure	Availability, latency, detection quality, evidence completeness, cryptographic cost, risk, resource use, and mission outcome.
Reconcile	Intended state against observed security, identity, application, cloud, cryptographic, data, and evidence records.

4.4 Operations and observability

Operations and observability must be implemented as an observable engineering mechanism, not a product label or maturity claim. Balance debugging, metrics, logs, crash dumps, performance, support, privacy, and evidence inside protected environments. The design should name authoritative inputs, identities, state transitions, dependencies, cryptographic or policy boundaries, limits, owners, telemetry, failure behavior, and acceptance criteria.

This guide traces the subject through the workload, TEE, evidence, verifier, appraisal, key release, protected computation, output, and relying-party decision chain. A configured integration, successful API response, established cryptographic session, green dashboard, or closed ticket is not sufficient proof. Intended state must be reconciled with applied state, negative tests, degraded behavior, raw evidence, recovery, and the resulting security or mission 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 authorized and prohibited behavior, inject realistic failures, measure convergence or recovery, and retain a tested rollback. Automation must stop safely when authority, freshness, identity, integrity, cryptographic state, or observed outcomes are ambiguous.

Troubleshooting locates the first divergence from the expected state machine or end-to-end transaction. Inspect every authority boundary and account for time, caching, eventual consistency, stale identity, incomplete collection, parser drift, asymmetric paths, encapsulation, supply-chain state, side channels, partial failure, and missing evidence. Closure requires causal explanation, reproducibility, validated restoration, residual limitations, and prevention or monitoring actions.

Engineering practices

Define authority, ownership, scope, dependencies, limits, and acceptance criteria for operations and observability.

Model expected state, trust boundaries, data flow, identity, and failure behavior before implementation.

Validate intended configuration, active state, applied enforcement, raw evidence, and end-to-end outcome independently.

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

Preserve policy, source, version, identity, telemetry, artifact, packet, log, time, and decision evidence.

Automate through typed contracts, least privilege, deterministic gates, canaries, bounded authority, and reversible changes.

Failure patterns

Treating operations and observability as a product feature instead of an end-to-end engineering system.

Relying on intended policy, rule code, or controller state without proving applied and operational behavior.

Ignoring data quality, identity, time, cryptographic lifecycle, supplier state, 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 result without negative tests, rollback, and recurrence monitoring.

Validation and evidence

Method	Expected evidence
Examine	Requirements, architecture, identities, data, algorithms, policy, configuration, ownership, and dependencies for operations and observability.
Observe	Active state, applied policy, raw events, telemetry, artifacts, cryptographic results, resources, and endpoint behavior.
Test	Expected behavior, prohibited behavior, malformed inputs, dependency loss, partial failure, scale, recovery, and rollback.
Measure	Availability, latency, detection quality, evidence completeness, cryptographic cost, risk, resource use, and mission outcome.
Reconcile	Intended state against observed security, identity, application, cloud, cryptographic, data, and evidence records.

Chapter 04 laboratory and review

Practical laboratory
Design a confidential AI inference service with attestation, key release, encrypted input, protected model, output policy, telemetry, and rollback.

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 confidential agent swarms, multi-GPU attestation, confidential RAN or edge workloads, and composable trusted devices.

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

Multi-party data collaboration and confidential services

Enable computation among parties that do not fully trust the infrastructure or each other.

Chapter operating position
Enable computation among parties that do not fully trust the infrastructure or each other.

5.1 Neutral execution and governance

Neutral execution and governance must be implemented as an observable engineering mechanism, not a product label or maturity claim. Define participants, code, data, outputs, policy, attestation, audit, operator limits, disputes, and termination. The design should name authoritative inputs, identities, state transitions, dependencies, cryptographic or policy boundaries, limits, owners, telemetry, failure behavior, and acceptance criteria.

This guide traces the subject through the workload, TEE, evidence, verifier, appraisal, key release, protected computation, output, and relying-party decision chain. A configured integration, successful API response, established cryptographic session, green dashboard, or closed ticket is not sufficient proof. Intended state must be reconciled with applied state, negative tests, degraded behavior, raw evidence, recovery, and the resulting security or mission 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 authorized and prohibited behavior, inject realistic failures, measure convergence or recovery, and retain a tested rollback. Automation must stop safely when authority, freshness, identity, integrity, cryptographic state, or observed outcomes are ambiguous.

Troubleshooting locates the first divergence from the expected state machine or end-to-end transaction. Inspect every authority boundary and account for time, caching, eventual consistency, stale identity, incomplete collection, parser drift, asymmetric paths, encapsulation, supply-chain state, side channels, partial failure, and missing evidence. Closure requires causal explanation, reproducibility, validated restoration, residual limitations, and prevention or monitoring actions.

Engineering practices

- Define authority, ownership, scope, dependencies, limits, and acceptance criteria for neutral execution and governance.
- Model expected state, trust boundaries, data flow, identity, and failure behavior before implementation.
- Validate intended configuration, active state, applied enforcement, raw evidence, and end-to-end outcome independently.
- Test normal, prohibited, degraded, failed, scale, recovery, and rollback conditions.
- Preserve policy, source, version, identity, telemetry, artifact, packet, log, time, and decision evidence.
- Automate through typed contracts, least privilege, deterministic gates, canaries, bounded authority, and reversible changes.

Failure patterns

- Treating neutral execution and governance as a product feature instead of an end-to-end engineering system.
- Relying on intended policy, rule code, or controller state without proving applied and operational behavior.
- Ignoring data quality, identity, time, cryptographic lifecycle, supplier state, 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 result without negative tests, rollback, and recurrence monitoring.

Validation and evidence

Method	Expected evidence
Examine	Requirements, architecture, identities, data, algorithms, policy, configuration, ownership, and dependencies for neutral execution and governance.
Observe	Active state, applied policy, raw events, telemetry, artifacts, cryptographic results, resources, and endpoint behavior.
Test	Expected behavior, prohibited behavior, malformed inputs, dependency loss, partial failure, scale, recovery, and rollback.
Measure	Availability, latency, detection quality, evidence completeness, cryptographic cost, risk, resource use, and mission outcome.
Reconcile	Intended state against observed security, identity, application, cloud, cryptographic, data, and evidence records.

5.2 Data clean rooms and analytics

Data clean rooms and analytics must be implemented as an observable engineering mechanism, not a product label or maturity claim. Control query templates, joins, minimum cohorts, output review, privacy budgets, export, retention, and evidence. The design should name authoritative inputs, identities, state transitions, dependencies, cryptographic or policy boundaries, limits, owners, telemetry, failure behavior, and acceptance criteria.

This guide traces the subject through the workload, TEE, evidence, verifier, appraisal, key release, protected computation, output, and relying-party decision chain. A configured integration, successful API response, established cryptographic session, green dashboard, or closed ticket is not sufficient proof. Intended state must be reconciled with applied state, negative tests, degraded behavior, raw evidence, recovery, and the resulting security or mission 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 authorized and prohibited behavior, inject realistic failures, measure convergence or recovery, and retain a tested rollback. Automation must stop safely when authority, freshness, identity, integrity, cryptographic state, or observed outcomes are ambiguous.

Troubleshooting locates the first divergence from the expected state machine or end-to-end transaction. Inspect every authority boundary and account for time, caching, eventual consistency, stale identity, incomplete collection, parser drift, asymmetric paths, encapsulation, supply-chain state, side channels, partial failure, and missing evidence. Closure requires causal explanation, reproducibility, validated restoration, residual limitations, and prevention or monitoring actions.

Engineering practices

- Define authority, ownership, scope, dependencies, limits, and acceptance criteria for data clean rooms and analytics.
- Model expected state, trust boundaries, data flow, identity, and failure behavior before implementation.
- Validate intended configuration, active state, applied enforcement, raw evidence, and end-to-end outcome independently.
- Test normal, prohibited, degraded, failed, scale, recovery, and rollback conditions.
- Preserve policy, source, version, identity, telemetry, artifact, packet, log, time, and decision evidence.
- Automate through typed contracts, least privilege, deterministic gates, canaries, bounded authority, and reversible changes.

Implementation sketch

```
attestation_policy:
  workload_digest: sha256:...
  signer: approved-release-identity
  platform:
    debug_disabled: true
    security_version_min: 7
  freshness:
    nonce_required: true
    max_age_seconds: 60
  decision:
    on_pass: release-model-decryption-key
    on_fail: deny-and-record
  evidence:
    retain: [eat-token, verifier-result, endorsements, reference-values]
  limitations:
    - availability-not-guaranteed
    - approved-code-can-still-leak-output
```

Failure patterns

Treating data clean rooms and analytics as a product feature instead of an end-to-end engineering system.

Relying on intended policy, rule code, or controller state without proving applied and operational behavior.

Ignoring data quality, identity, time, cryptographic lifecycle, supplier state, 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 result without negative tests, rollback, and recurrence monitoring.

Validation and evidence

Method	Expected evidence
Examine	Requirements, architecture, identities, data, algorithms, policy, configuration, ownership, and dependencies for data clean rooms and analytics.
Observe	Active state, applied policy, raw events, telemetry, artifacts, cryptographic results, resources, and endpoint behavior.
Test	Expected behavior, prohibited behavior, malformed inputs, dependency loss, partial failure, scale, recovery, and rollback.
Measure	Availability, latency, detection quality, evidence completeness, cryptographic cost, risk, resource use, and mission outcome.
Reconcile	Intended state against observed security, identity, application, cloud, cryptographic, data, and evidence records.

5.3 Federated and cross-organization systems

Federated and cross-organization systems must be implemented as an observable engineering mechanism, not a product label or maturity claim. Coordinate identities, policies, keys, jurisdictions, schemas, updates, and trust across parties. The design should name authoritative inputs, identities, state transitions, dependencies, cryptographic or policy boundaries, limits, owners, telemetry, failure behavior, and acceptance criteria.

This guide traces the subject through the workload, TEE, evidence, verifier, appraisal, key release, protected computation, output, and relying-party decision chain. A configured integration, successful API response, established cryptographic session, green dashboard, or closed ticket is not sufficient proof. Intended state must be reconciled with applied state, negative tests, degraded behavior, raw evidence, recovery, and the resulting security or mission 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 authorized and prohibited behavior, inject realistic failures, measure convergence or recovery, and retain a tested rollback. Automation must stop safely when authority, freshness, identity, integrity, cryptographic state, or observed outcomes are ambiguous.

Troubleshooting locates the first divergence from the expected state machine or end-to-end transaction. Inspect every authority boundary and account for time, caching, eventual consistency, stale identity, incomplete collection, parser drift, asymmetric

paths, encapsulation, supply-chain state, side channels, partial failure, and missing evidence. Closure requires causal explanation, reproducibility, validated restoration, residual limitations, and prevention or monitoring actions.

Engineering practices

- Define authority, ownership, scope, dependencies, limits, and acceptance criteria for federated and cross-organization systems.
- Model expected state, trust boundaries, data flow, identity, and failure behavior before implementation.
- Validate intended configuration, active state, applied enforcement, raw evidence, and end-to-end outcome independently.
- Test normal, prohibited, degraded, failed, scale, recovery, and rollback conditions.
- Preserve policy, source, version, identity, telemetry, artifact, packet, log, time, and decision evidence.
- Automate through typed contracts, least privilege, deterministic gates, canaries, bounded authority, and reversible changes.

Failure patterns

- Treating federated and cross-organization systems as a product feature instead of an end-to-end engineering system.
- Relying on intended policy, rule code, or controller state without proving applied and operational behavior.
- Ignoring data quality, identity, time, cryptographic lifecycle, supplier state, 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 result without negative tests, rollback, and recurrence monitoring.

Validation and evidence

Method	Expected evidence
Examine	Requirements, architecture, identities, data, algorithms, policy, configuration, ownership, and dependencies for federated and cross-organization systems.
Observe	Active state, applied policy, raw events, telemetry, artifacts, cryptographic results, resources, and endpoint behavior.
Test	Expected behavior, prohibited behavior, malformed inputs, dependency loss, partial failure, scale, recovery, and rollback.
Measure	Availability, latency, detection quality, evidence completeness, cryptographic cost, risk, resource use, and mission outcome.
Reconcile	Intended state against observed security, identity, application, cloud, cryptographic, data, and evidence records.

5.4 Limitations and collusion

Limitations and collusion must be implemented as an observable engineering mechanism, not a product label or maturity claim. Analyze malicious workloads, approved-but-harmful queries, side channels, operator denial, participant collusion, and output leakage. The design should name authoritative inputs, identities, state transitions, dependencies, cryptographic or policy boundaries, limits, owners, telemetry, failure behavior, and acceptance criteria.

This guide traces the subject through the workload, TEE, evidence, verifier, appraisal, key release, protected computation, output, and relying-party decision chain. A configured integration, successful API response, established cryptographic session, green dashboard, or closed ticket is not sufficient proof. Intended state must be reconciled with applied state, negative tests, degraded behavior, raw evidence, recovery, and the resulting security or mission 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 authorized and prohibited behavior, inject realistic failures, measure convergence or recovery, and retain a tested rollback. Automation must stop safely when authority, freshness, identity, integrity, cryptographic state, or observed outcomes are ambiguous.

Troubleshooting locates the first divergence from the expected state machine or end-to-end transaction. Inspect every authority boundary and account for time, caching, eventual consistency, stale identity, incomplete collection, parser drift, asymmetric paths, encapsulation, supply-chain state, side channels, partial failure, and missing evidence. Closure requires causal explanation, reproducibility, validated restoration, residual limitations, and prevention or monitoring actions.

Engineering practices

Define authority, ownership, scope, dependencies, limits, and acceptance criteria for limitations and collusion.

Model expected state, trust boundaries, data flow, identity, and failure behavior before implementation.

Validate intended configuration, active state, applied enforcement, raw evidence, and end-to-end outcome independently.

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

Preserve policy, source, version, identity, telemetry, artifact, packet, log, time, and decision evidence.

Automate through typed contracts, least privilege, deterministic gates, canaries, bounded authority, and reversible changes.

Failure patterns

Treating limitations and collusion as a product feature instead of an end-to-end engineering system.

Relying on intended policy, rule code, or controller state without proving applied and operational behavior.

Ignoring data quality, identity, time, cryptographic lifecycle, supplier state, 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 result without negative tests, rollback, and recurrence monitoring.

Validation and evidence

Method	Expected evidence
Examine	Requirements, architecture, identities, data, algorithms, policy, configuration, ownership, and dependencies for limitations and collusion.
Observe	Active state, applied policy, raw events, telemetry, artifacts, cryptographic results, resources, and endpoint behavior.
Test	Expected behavior, prohibited behavior, malformed inputs, dependency loss, partial failure, scale, recovery, and rollback.
Measure	Availability, latency, detection quality, evidence completeness, cryptographic cost, risk, resource use, and mission outcome.
Reconcile	Intended state against observed security, identity, application, cloud, cryptographic, data, and evidence records.

Chapter 05 laboratory and review

Practical laboratory
Design a two-party confidential analytics service and test unauthorized code, disallowed query, stale attestation, excessive output, and participant withdrawal.

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 verifiable data marketplaces, sovereign collaboration, confidential federated learning, and decentralized attestation.

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

Fully homomorphic encryption engineering

Evaluate computation on encrypted data according to scheme, parameters, workload, and operational cost.

Chapter operating position
Evaluate computation on encrypted data according to scheme, parameters, workload, and operational cost.

6.1 FHE concepts and schemes

FHE concepts and schemes must be implemented as an observable engineering mechanism, not a product label or maturity claim. Understand keys, ciphertexts, plaintext spaces, noise, levels, bootstrapping, exact and approximate arithmetic, and security assumptions. The design should name authoritative inputs, identities, state transitions, dependencies, cryptographic or policy boundaries, limits, owners, telemetry, failure behavior, and acceptance criteria.

This guide traces the subject through the workload, TEE, evidence, verifier, appraisal, key release, protected computation, output, and relying-party decision chain. A configured integration, successful API response, established cryptographic session, green dashboard, or closed ticket is not sufficient proof. Intended state must be reconciled with applied state, negative tests, degraded behavior, raw evidence, recovery, and the resulting security or mission 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 authorized and prohibited behavior, inject realistic failures, measure convergence or recovery, and retain a tested rollback. Automation must stop safely when authority, freshness, identity, integrity, cryptographic state, or observed outcomes are ambiguous.

Troubleshooting locates the first divergence from the expected state machine or end-to-end transaction. Inspect every authority boundary and account for time, caching, eventual consistency, stale identity, incomplete collection, parser drift, asymmetric paths, encapsulation, supply-chain state, side channels, partial failure, and missing evidence. Closure requires causal explanation, reproducibility, validated restoration, residual limitations, and prevention or monitoring actions.

Engineering practices

- Define authority, ownership, scope, dependencies, limits, and acceptance criteria for fhe concepts and schemes.
- Model expected state, trust boundaries, data flow, identity, and failure behavior before implementation.
- Validate intended configuration, active state, applied enforcement, raw evidence, and end-to-end outcome independently.
- Test normal, prohibited, degraded, failed, scale, recovery, and rollback conditions.
- Preserve policy, source, version, identity, telemetry, artifact, packet, log, time, and decision evidence.
- Automate through typed contracts, least privilege, deterministic gates, canaries, bounded authority, and reversible changes.

Failure patterns

- Treating fhe concepts and schemes as a product feature instead of an end-to-end engineering system.
- Relying on intended policy, rule code, or controller state without proving applied and operational behavior.
- Ignoring data quality, identity, time, cryptographic lifecycle, supplier state, 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 result without negative tests, rollback, and recurrence monitoring.

Validation and evidence

Method	Expected evidence
Examine	Requirements, architecture, identities, data, algorithms, policy, configuration, ownership, and dependencies for fhe concepts and schemes.
Observe	Active state, applied policy, raw events, telemetry, artifacts, cryptographic results, resources, and endpoint behavior.
Test	Expected behavior, prohibited behavior, malformed inputs, dependency loss, partial failure, scale, recovery, and rollback.
Measure	Availability, latency, detection quality, evidence completeness, cryptographic cost, risk, resource use, and mission outcome.
Reconcile	Intended state against observed security, identity, application, cloud, cryptographic, data, and evidence records.

6.2 BFV, BGV, CKKS, and TFHE-style models

BFV, BGV, CKKS, and TFHE-style models must be implemented as an observable engineering mechanism, not a product label or maturity claim. Compare integer or modular arithmetic, approximate numeric computation, Boolean or gate operations, packing, and workloads. The design should name authoritative inputs, identities, state transitions, dependencies, cryptographic or policy boundaries, limits, owners, telemetry, failure behavior, and acceptance criteria.

This guide traces the subject through the workload, TEE, evidence, verifier, appraisal, key release, protected computation, output, and relying-party decision chain. A configured integration, successful API response, established cryptographic session, green dashboard, or closed ticket is not sufficient proof. Intended state must be reconciled with applied state, negative tests, degraded behavior, raw evidence, recovery, and the resulting security or mission 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 authorized and prohibited behavior, inject realistic failures, measure convergence or recovery, and retain a tested rollback. Automation must stop safely when authority, freshness, identity, integrity, cryptographic state, or observed outcomes are ambiguous.

Troubleshooting locates the first divergence from the expected state machine or end-to-end transaction. Inspect every authority boundary and account for time, caching, eventual consistency, stale identity, incomplete collection, parser drift, asymmetric paths, encapsulation, supply-chain state, side channels, partial failure, and missing evidence. Closure requires causal explanation, reproducibility, validated restoration, residual limitations, and prevention or monitoring actions.

Engineering practices

- Define authority, ownership, scope, dependencies, limits, and acceptance criteria for bfv, bgv, ckks, and tfhe-style models.
- Model expected state, trust boundaries, data flow, identity, and failure behavior before implementation.
- Validate intended configuration, active state, applied enforcement, raw evidence, and end-to-end outcome independently.
- Test normal, prohibited, degraded, failed, scale, recovery, and rollback conditions.
- Preserve policy, source, version, identity, telemetry, artifact, packet, log, time, and decision evidence.
- Automate through typed contracts, least privilege, deterministic gates, canaries, bounded authority, and reversible changes.

Implementation sketch

```
attestation_policy:
  workload_digest: sha256:...
  signer: approved-release-identity
  platform:
    debug_disabled: true
    security_version_min: 7
  freshness:
    nonce_required: true
    max_age_seconds: 60
  decision:
    on_pass: release-model-decryption-key
    on_fail: deny-and-record
  evidence:
    retain: [eat-token, verifier-result, endorsements, reference-values]
  limitations:
    - availability-not-guaranteed
    - approved-code-can-still-leak-output
```

Failure patterns

Treating bfv, bgv, ckks, and tfhe-style models as a product feature instead of an end-to-end engineering system.

Relying on intended policy, rule code, or controller state without proving applied and operational behavior.

Ignoring data quality, identity, time, cryptographic lifecycle, supplier state, 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 result without negative tests, rollback, and recurrence monitoring.

Validation and evidence

Method	Expected evidence
Examine	Requirements, architecture, identities, data, algorithms, policy, configuration, ownership, and dependencies for bfv, bgv, ckks, and tfhe-style models.
Observe	Active state, applied policy, raw events, telemetry, artifacts, cryptographic results, resources, and endpoint behavior.
Test	Expected behavior, prohibited behavior, malformed inputs, dependency loss, partial failure, scale, recovery, and rollback.
Measure	Availability, latency, detection quality, evidence completeness, cryptographic cost, risk, resource use, and mission outcome.
Reconcile	Intended state against observed security, identity, application, cloud, cryptographic, data, and evidence records.

6.3 Circuit and parameter engineering

Circuit and parameter engineering must be implemented as an observable engineering mechanism, not a product label or maturity claim. Select depth, modulus, ring dimension, scale, precision, rotations, keys, batching, and security according to the computation. The design should name authoritative inputs, identities, state transitions, dependencies, cryptographic or policy boundaries, limits, owners, telemetry, failure behavior, and acceptance criteria.

This guide traces the subject through the workload, TEE, evidence, verifier, appraisal, key release, protected computation, output, and relying-party decision chain. A configured integration, successful API response, established cryptographic session, green dashboard, or closed ticket is not sufficient proof. Intended state must be reconciled with applied state, negative tests, degraded behavior, raw evidence, recovery, and the resulting security or mission 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 authorized and prohibited behavior, inject realistic failures, measure convergence or recovery, and retain a tested rollback. Automation must stop safely when authority, freshness, identity, integrity, cryptographic state, or observed outcomes are ambiguous.

Troubleshooting locates the first divergence from the expected state machine or end-to-end transaction. Inspect every authority boundary and account for time, caching, eventual consistency, stale identity, incomplete collection, parser drift, asymmetric

paths, encapsulation, supply-chain state, side channels, partial failure, and missing evidence. Closure requires causal explanation, reproducibility, validated restoration, residual limitations, and prevention or monitoring actions.

Engineering practices

- Define authority, ownership, scope, dependencies, limits, and acceptance criteria for circuit and parameter engineering.
- Model expected state, trust boundaries, data flow, identity, and failure behavior before implementation.
- Validate intended configuration, active state, applied enforcement, raw evidence, and end-to-end outcome independently.
- Test normal, prohibited, degraded, failed, scale, recovery, and rollback conditions.
- Preserve policy, source, version, identity, telemetry, artifact, packet, log, time, and decision evidence.
- Automate through typed contracts, least privilege, deterministic gates, canaries, bounded authority, and reversible changes.

Failure patterns

- Treating circuit and parameter engineering as a product feature instead of an end-to-end engineering system.
- Relying on intended policy, rule code, or controller state without proving applied and operational behavior.
- Ignoring data quality, identity, time, cryptographic lifecycle, supplier state, 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 result without negative tests, rollback, and recurrence monitoring.

Validation and evidence

Method	Expected evidence
Examine	Requirements, architecture, identities, data, algorithms, policy, configuration, ownership, and dependencies for circuit and parameter engineering.
Observe	Active state, applied policy, raw events, telemetry, artifacts, cryptographic results, resources, and endpoint behavior.
Test	Expected behavior, prohibited behavior, malformed inputs, dependency loss, partial failure, scale, recovery, and rollback.
Measure	Availability, latency, detection quality, evidence completeness, cryptographic cost, risk, resource use, and mission outcome.
Reconcile	Intended state against observed security, identity, application, cloud, cryptographic, data, and evidence records.

6.4 Performance, correctness, and deployment

Performance, correctness, and deployment must be implemented as an observable engineering mechanism, not a product label or maturity claim. Measure latency, throughput, memory, ciphertext expansion, communication, precision error, hardware acceleration, and key custody. The design should name authoritative inputs, identities, state transitions, dependencies, cryptographic or policy boundaries, limits, owners, telemetry, failure behavior, and acceptance criteria.

This guide traces the subject through the workload, TEE, evidence, verifier, appraisal, key release, protected computation, output, and relying-party decision chain. A configured integration, successful API response, established cryptographic session, green dashboard, or closed ticket is not sufficient proof. Intended state must be reconciled with applied state, negative tests, degraded behavior, raw evidence, recovery, and the resulting security or mission 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 authorized and prohibited behavior, inject realistic failures, measure convergence or recovery, and retain a tested rollback. Automation must stop safely when authority, freshness, identity, integrity, cryptographic state, or observed outcomes are ambiguous.

Troubleshooting locates the first divergence from the expected state machine or end-to-end transaction. Inspect every authority boundary and account for time, caching, eventual consistency, stale identity, incomplete collection, parser drift, asymmetric paths, encapsulation, supply-chain state, side channels, partial failure, and missing evidence. Closure requires causal explanation, reproducibility, validated restoration, residual limitations, and prevention or monitoring actions.

Engineering practices

Define authority, ownership, scope, dependencies, limits, and acceptance criteria for performance, correctness, and deployment.

Model expected state, trust boundaries, data flow, identity, and failure behavior before implementation.

Validate intended configuration, active state, applied enforcement, raw evidence, and end-to-end outcome independently.

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

Preserve policy, source, version, identity, telemetry, artifact, packet, log, time, and decision evidence.

Automate through typed contracts, least privilege, deterministic gates, canaries, bounded authority, and reversible changes.

Failure patterns

Treating performance, correctness, and deployment as a product feature instead of an end-to-end engineering system.

Relying on intended policy, rule code, or controller state without proving applied and operational behavior.

Ignoring data quality, identity, time, cryptographic lifecycle, supplier state, 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 result without negative tests, rollback, and recurrence monitoring.

Validation and evidence

Method	Expected evidence
Examine	Requirements, architecture, identities, data, algorithms, policy, configuration, ownership, and dependencies for performance, correctness, and deployment.
Observe	Active state, applied policy, raw events, telemetry, artifacts, cryptographic results, resources, and endpoint behavior.
Test	Expected behavior, prohibited behavior, malformed inputs, dependency loss, partial failure, scale, recovery, and rollback.
Measure	Availability, latency, detection quality, evidence completeness, cryptographic cost, risk, resource use, and mission outcome.
Reconcile	Intended state against observed security, identity, application, cloud, cryptographic, data, and evidence records.

Chapter 06 laboratory and review

Practical laboratory
Implement or model an encrypted statistical or inference workload, select parameters, measure error and performance, and compare with a TEE design.

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 compiler-assisted FHE, encrypted AI inference, multi-key FHE, hardware accelerators, and evolving NIST PEC work.

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

Zero-knowledge proof engineering

Prove statements or knowledge without revealing the witness beyond intended leakage.

Chapter operating position
Prove statements or knowledge without revealing the witness beyond intended leakage.

7.1 Statements, instances, witnesses, and relations

Statements, instances, witnesses, and relations must be implemented as an observable engineering mechanism, not a product label or maturity claim. Define the exact claim, public input, secret witness, computation, and what is or is not revealed. The design should name authoritative inputs, identities, state transitions, dependencies, cryptographic or policy boundaries, limits, owners, telemetry, failure behavior, and acceptance criteria.

This guide traces the subject through the workload, TEE, evidence, verifier, appraisal, key release, protected computation, output, and relying-party decision chain. A configured integration, successful API response, established cryptographic session, green dashboard, or closed ticket is not sufficient proof. Intended state must be reconciled with applied state, negative tests, degraded behavior, raw evidence, recovery, and the resulting security or mission 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 authorized and prohibited behavior, inject realistic failures, measure convergence or recovery, and retain a tested rollback. Automation must stop safely when authority, freshness, identity, integrity, cryptographic state, or observed outcomes are ambiguous.

Troubleshooting locates the first divergence from the expected state machine or end-to-end transaction. Inspect every authority boundary and account for time, caching, eventual consistency, stale identity, incomplete collection, parser drift, asymmetric paths, encapsulation, supply-chain state, side channels, partial failure, and missing evidence. Closure requires causal explanation, reproducibility, validated restoration, residual limitations, and prevention or monitoring actions.

Engineering practices

- Define authority, ownership, scope, dependencies, limits, and acceptance criteria for statements, instances, witnesses, and relations.
- Model expected state, trust boundaries, data flow, identity, and failure behavior before implementation.
- Validate intended configuration, active state, applied enforcement, raw evidence, and end-to-end outcome independently.
- Test normal, prohibited, degraded, failed, scale, recovery, and rollback conditions.
- Preserve policy, source, version, identity, telemetry, artifact, packet, log, time, and decision evidence.
- Automate through typed contracts, least privilege, deterministic gates, canaries, bounded authority, and reversible changes.

Failure patterns

- Treating statements, instances, witnesses, and relations as a product feature instead of an end-to-end engineering system.
- Relying on intended policy, rule code, or controller state without proving applied and operational behavior.
- Ignoring data quality, identity, time, cryptographic lifecycle, supplier state, 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 result without negative tests, rollback, and recurrence monitoring.

Validation and evidence

Method	Expected evidence
Examine	Requirements, architecture, identities, data, algorithms, policy, configuration, ownership, and dependencies for statements, instances, witnesses, and relations.
Observe	Active state, applied policy, raw events, telemetry, artifacts, cryptographic results, resources, and endpoint behavior.
Test	Expected behavior, prohibited behavior, malformed inputs, dependency loss, partial failure, scale, recovery, and rollback.
Measure	Availability, latency, detection quality, evidence completeness, cryptographic cost, risk, resource use, and mission outcome.
Reconcile	Intended state against observed security, identity, application, cloud, cryptographic, data, and evidence records.

7.2 Completeness, soundness, and zero knowledge

Completeness, soundness, and zero knowledge must be implemented as an observable engineering mechanism, not a product label or maturity claim. Understand security properties, proof versus argument, knowledge extraction, assumptions, and composition. The design should name authoritative inputs, identities, state transitions, dependencies, cryptographic or policy boundaries, limits, owners, telemetry, failure behavior, and acceptance criteria.

This guide traces the subject through the workload, TEE, evidence, verifier, appraisal, key release, protected computation, output, and relying-party decision chain. A configured integration, successful API response, established cryptographic session, green dashboard, or closed ticket is not sufficient proof. Intended state must be reconciled with applied state, negative tests, degraded behavior, raw evidence, recovery, and the resulting security or mission 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 authorized and prohibited behavior, inject realistic failures, measure convergence or recovery, and retain a tested rollback. Automation must stop safely when authority, freshness, identity, integrity, cryptographic state, or observed outcomes are ambiguous.

Troubleshooting locates the first divergence from the expected state machine or end-to-end transaction. Inspect every authority boundary and account for time, caching, eventual consistency, stale identity, incomplete collection, parser drift, asymmetric paths, encapsulation, supply-chain state, side channels, partial failure, and missing evidence. Closure requires causal explanation, reproducibility, validated restoration, residual limitations, and prevention or monitoring actions.

Engineering practices

- Define authority, ownership, scope, dependencies, limits, and acceptance criteria for completeness, soundness, and zero knowledge.
- Model expected state, trust boundaries, data flow, identity, and failure behavior before implementation.
- Validate intended configuration, active state, applied enforcement, raw evidence, and end-to-end outcome independently.
- Test normal, prohibited, degraded, failed, scale, recovery, and rollback conditions.
- Preserve policy, source, version, identity, telemetry, artifact, packet, log, time, and decision evidence.
- Automate through typed contracts, least privilege, deterministic gates, canaries, bounded authority, and reversible changes.

Implementation sketch

```
attestation_policy:
  workload_digest: sha256:...
  signer: approved-release-identity
  platform:
    debug_disabled: true
    security_version_min: 7
  freshness:
    nonce_required: true
    max_age_seconds: 60
  decision:
    on_pass: release-model-decryption-key
    on_fail: deny-and-record
  evidence:
    retain: [eat-token, verifier-result, endorsements, reference-values]
  limitations:
    - availability-not-guaranteed
    - approved-code-can-still-leak-output
```

Failure patterns

Treating completeness, soundness, and zero knowledge as a product feature instead of an end-to-end engineering system.

Relying on intended policy, rule code, or controller state without proving applied and operational behavior.

Ignoring data quality, identity, time, cryptographic lifecycle, supplier state, 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 result without negative tests, rollback, and recurrence monitoring.

Validation and evidence

Method	Expected evidence
Examine	Requirements, architecture, identities, data, algorithms, policy, configuration, ownership, and dependencies for completeness, soundness, and zero knowledge.
Observe	Active state, applied policy, raw events, telemetry, artifacts, cryptographic results, resources, and endpoint behavior.
Test	Expected behavior, prohibited behavior, malformed inputs, dependency loss, partial failure, scale, recovery, and rollback.
Measure	Availability, latency, detection quality, evidence completeness, cryptographic cost, risk, resource use, and mission outcome.
Reconcile	Intended state against observed security, identity, application, cloud, cryptographic, data, and evidence records.

7.3 Proof-system architecture

Proof-system architecture must be implemented as an observable engineering mechanism, not a product label or maturity claim. Compare interactive and noninteractive systems, trusted or transparent setup, arithmetic circuits, commitments, transcripts, and Fiat-Shamir use. The design should name authoritative inputs, identities, state transitions, dependencies, cryptographic or policy boundaries, limits, owners, telemetry, failure behavior, and acceptance criteria.

This guide traces the subject through the workload, TEE, evidence, verifier, appraisal, key release, protected computation, output, and relying-party decision chain. A configured integration, successful API response, established cryptographic session, green dashboard, or closed ticket is not sufficient proof. Intended state must be reconciled with applied state, negative tests, degraded behavior, raw evidence, recovery, and the resulting security or mission 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 authorized and prohibited behavior, inject realistic failures, measure convergence or recovery, and retain a tested rollback. Automation must stop safely when authority, freshness, identity, integrity, cryptographic state, or observed outcomes are ambiguous.

Troubleshooting locates the first divergence from the expected state machine or end-to-end transaction. Inspect every authority boundary and account for time, caching, eventual consistency, stale identity, incomplete collection, parser drift, asymmetric

paths, encapsulation, supply-chain state, side channels, partial failure, and missing evidence. Closure requires causal explanation, reproducibility, validated restoration, residual limitations, and prevention or monitoring actions.

Engineering practices

- Define authority, ownership, scope, dependencies, limits, and acceptance criteria for proof-system architecture.
- Model expected state, trust boundaries, data flow, identity, and failure behavior before implementation.
- Validate intended configuration, active state, applied enforcement, raw evidence, and end-to-end outcome independently.
- Test normal, prohibited, degraded, failed, scale, recovery, and rollback conditions.
- Preserve policy, source, version, identity, telemetry, artifact, packet, log, time, and decision evidence.
- Automate through typed contracts, least privilege, deterministic gates, canaries, bounded authority, and reversible changes.

Failure patterns

- Treating proof-system architecture as a product feature instead of an end-to-end engineering system.
- Relying on intended policy, rule code, or controller state without proving applied and operational behavior.
- Ignoring data quality, identity, time, cryptographic lifecycle, supplier state, 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 result without negative tests, rollback, and recurrence monitoring.

Validation and evidence

Method	Expected evidence
Examine	Requirements, architecture, identities, data, algorithms, policy, configuration, ownership, and dependencies for proof-system architecture.
Observe	Active state, applied policy, raw events, telemetry, artifacts, cryptographic results, resources, and endpoint behavior.
Test	Expected behavior, prohibited behavior, malformed inputs, dependency loss, partial failure, scale, recovery, and rollback.
Measure	Availability, latency, detection quality, evidence completeness, cryptographic cost, risk, resource use, and mission outcome.
Reconcile	Intended state against observed security, identity, application, cloud, cryptographic, data, and evidence records.

7.4 Deployment and assurance

Deployment and assurance must be implemented as an observable engineering mechanism, not a product label or maturity claim. Measure prover time, verifier time, proof size, memory, recursion, setup, key lifecycle, implementation risk, and side channels. The design should name authoritative inputs, identities, state transitions, dependencies, cryptographic or policy boundaries, limits, owners, telemetry, failure behavior, and acceptance criteria.

This guide traces the subject through the workload, TEE, evidence, verifier, appraisal, key release, protected computation, output, and relying-party decision chain. A configured integration, successful API response, established cryptographic session, green dashboard, or closed ticket is not sufficient proof. Intended state must be reconciled with applied state, negative tests, degraded behavior, raw evidence, recovery, and the resulting security or mission 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 authorized and prohibited behavior, inject realistic failures, measure convergence or recovery, and retain a tested rollback. Automation must stop safely when authority, freshness, identity, integrity, cryptographic state, or observed outcomes are ambiguous.

Troubleshooting locates the first divergence from the expected state machine or end-to-end transaction. Inspect every authority boundary and account for time, caching, eventual consistency, stale identity, incomplete collection, parser drift, asymmetric paths, encapsulation, supply-chain state, side channels, partial failure, and missing evidence. Closure requires causal explanation, reproducibility, validated restoration, residual limitations, and prevention or monitoring actions.

Engineering practices

Define authority, ownership, scope, dependencies, limits, and acceptance criteria for deployment and assurance.

Model expected state, trust boundaries, data flow, identity, and failure behavior before implementation.

Validate intended configuration, active state, applied enforcement, raw evidence, and end-to-end outcome independently.

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

Preserve policy, source, version, identity, telemetry, artifact, packet, log, time, and decision evidence.

Automate through typed contracts, least privilege, deterministic gates, canaries, bounded authority, and reversible changes.

Failure patterns

Treating deployment and assurance as a product feature instead of an end-to-end engineering system.

Relying on intended policy, rule code, or controller state without proving applied and operational behavior.

Ignoring data quality, identity, time, cryptographic lifecycle, supplier state, 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 result without negative tests, rollback, and recurrence monitoring.

Validation and evidence

Method	Expected evidence
Examine	Requirements, architecture, identities, data, algorithms, policy, configuration, ownership, and dependencies for deployment and assurance.
Observe	Active state, applied policy, raw events, telemetry, artifacts, cryptographic results, resources, and endpoint behavior.
Test	Expected behavior, prohibited behavior, malformed inputs, dependency loss, partial failure, scale, recovery, and rollback.
Measure	Availability, latency, detection quality, evidence completeness, cryptographic cost, risk, resource use, and mission outcome.
Reconcile	Intended state against observed security, identity, application, cloud, cryptographic, data, and evidence records.

Chapter 07 laboratory and review

Practical laboratory
Design a ZKP for an authorization or compliance statement, explicitly defining witness, public data, setup, leakage, verification, and failure.

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 recursive proofs, proof-carrying data, verifiable AI inference, selective disclosure, and active CFRG research.

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

Architecture selection, assurance, and frontier adoption

Choose and combine advanced assurance tools according to concrete requirements and maturity.

Chapter operating position
Choose and combine advanced assurance tools according to concrete requirements and maturity.

8.1 TEE, FHE, ZKP, MPC, and conventional controls

TEE, FHE, ZKP, MPC, and conventional controls must be implemented as an observable engineering mechanism, not a product label or maturity claim. Compare trust, confidentiality, integrity, verifiability, interactivity, performance, maturity, and operational complexity. The design should name authoritative inputs, identities, state transitions, dependencies, cryptographic or policy boundaries, limits, owners, telemetry, failure behavior, and acceptance criteria.

This guide traces the subject through the workload, TEE, evidence, verifier, appraisal, key release, protected computation, output, and relying-party decision chain. A configured integration, successful API response, established cryptographic session, green dashboard, or closed ticket is not sufficient proof. Intended state must be reconciled with applied state, negative tests, degraded behavior, raw evidence, recovery, and the resulting security or mission 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 authorized and prohibited behavior, inject realistic failures, measure convergence or recovery, and retain a tested rollback. Automation must stop safely when authority, freshness, identity, integrity, cryptographic state, or observed outcomes are ambiguous.

Troubleshooting locates the first divergence from the expected state machine or end-to-end transaction. Inspect every authority boundary and account for time, caching, eventual consistency, stale identity, incomplete collection, parser drift, asymmetric paths, encapsulation, supply-chain state, side channels, partial failure, and missing evidence. Closure requires causal explanation, reproducibility, validated restoration, residual limitations, and prevention or monitoring actions.

Engineering practices

- Define authority, ownership, scope, dependencies, limits, and acceptance criteria for tee, fhe, zkp, mpc, and conventional controls.
- Model expected state, trust boundaries, data flow, identity, and failure behavior before implementation.
- Validate intended configuration, active state, applied enforcement, raw evidence, and end-to-end outcome independently.
- Test normal, prohibited, degraded, failed, scale, recovery, and rollback conditions.
- Preserve policy, source, version, identity, telemetry, artifact, packet, log, time, and decision evidence.
- Automate through typed contracts, least privilege, deterministic gates, canaries, bounded authority, and reversible changes.

Failure patterns

- Treating tee, fhe, zkp, mpc, and conventional controls as a product feature instead of an end-to-end engineering system.
- Relying on intended policy, rule code, or controller state without proving applied and operational behavior.
- Ignoring data quality, identity, time, cryptographic lifecycle, supplier state, 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 result without negative tests, rollback, and recurrence monitoring.

Validation and evidence

Method	Expected evidence
Examine	Requirements, architecture, identities, data, algorithms, policy, configuration, ownership, and dependencies for tee, fhe, zkp, mpc, and conventional controls.
Observe	Active state, applied policy, raw events, telemetry, artifacts, cryptographic results, resources, and endpoint behavior.
Test	Expected behavior, prohibited behavior, malformed inputs, dependency loss, partial failure, scale, recovery, and rollback.
Measure	Availability, latency, detection quality, evidence completeness, cryptographic cost, risk, resource use, and mission outcome.
Reconcile	Intended state against observed security, identity, application, cloud, cryptographic, data, and evidence records.

8.2 Composition and hybrid architectures

Composition and hybrid architectures must be implemented as an observable engineering mechanism, not a product label or maturity claim. Use TEEs with ZKP, FHE with MPC, attested key services, encrypted storage, secure channels, and application policy deliberately. The design should name authoritative inputs, identities, state transitions, dependencies, cryptographic or policy boundaries, limits, owners, telemetry, failure behavior, and acceptance criteria.

This guide traces the subject through the workload, TEE, evidence, verifier, appraisal, key release, protected computation, output, and relying-party decision chain. A configured integration, successful API response, established cryptographic session, green dashboard, or closed ticket is not sufficient proof. Intended state must be reconciled with applied state, negative tests, degraded behavior, raw evidence, recovery, and the resulting security or mission 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 authorized and prohibited behavior, inject realistic failures, measure convergence or recovery, and retain a tested rollback. Automation must stop safely when authority, freshness, identity, integrity, cryptographic state, or observed outcomes are ambiguous.

Troubleshooting locates the first divergence from the expected state machine or end-to-end transaction. Inspect every authority boundary and account for time, caching, eventual consistency, stale identity, incomplete collection, parser drift, asymmetric paths, encapsulation, supply-chain state, side channels, partial failure, and missing evidence. Closure requires causal explanation, reproducibility, validated restoration, residual limitations, and prevention or monitoring actions.

Engineering practices

- Define authority, ownership, scope, dependencies, limits, and acceptance criteria for composition and hybrid architectures.
- Model expected state, trust boundaries, data flow, identity, and failure behavior before implementation.
- Validate intended configuration, active state, applied enforcement, raw evidence, and end-to-end outcome independently.
- Test normal, prohibited, degraded, failed, scale, recovery, and rollback conditions.
- Preserve policy, source, version, identity, telemetry, artifact, packet, log, time, and decision evidence.
- Automate through typed contracts, least privilege, deterministic gates, canaries, bounded authority, and reversible changes.

Implementation sketch

```
attestation_policy:
  workload_digest: sha256:...
  signer: approved-release-identity
  platform:
    debug_disabled: true
    security_version_min: 7
  freshness:
    nonce_required: true
    max_age_seconds: 60
  decision:
    on_pass: release-model-decryption-key
    on_fail: deny-and-record
  evidence:
    retain: [eat-token, verifier-result, endorsements, reference-values]
  limitations:
    - availability-not-guaranteed
    - approved-code-can-still-leak-output
```

Failure patterns

Treating composition and hybrid architectures as a product feature instead of an end-to-end engineering system.

Relying on intended policy, rule code, or controller state without proving applied and operational behavior.

Ignoring data quality, identity, time, cryptographic lifecycle, supplier state, 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 result without negative tests, rollback, and recurrence monitoring.

Validation and evidence

Method	Expected evidence
Examine	Requirements, architecture, identities, data, algorithms, policy, configuration, ownership, and dependencies for composition and hybrid architectures.
Observe	Active state, applied policy, raw events, telemetry, artifacts, cryptographic results, resources, and endpoint behavior.
Test	Expected behavior, prohibited behavior, malformed inputs, dependency loss, partial failure, scale, recovery, and rollback.
Measure	Availability, latency, detection quality, evidence completeness, cryptographic cost, risk, resource use, and mission outcome.
Reconcile	Intended state against observed security, identity, application, cloud, cryptographic, data, and evidence records.

8.3 Testing and evidence

Testing and evidence must be implemented as an observable engineering mechanism, not a product label or maturity claim. Validate attestation, reference values, isolation, side channels, cryptography, correctness, outputs, updates, rollback, and incident response. The design should name authoritative inputs, identities, state transitions, dependencies, cryptographic or policy boundaries, limits, owners, telemetry, failure behavior, and acceptance criteria.

This guide traces the subject through the workload, TEE, evidence, verifier, appraisal, key release, protected computation, output, and relying-party decision chain. A configured integration, successful API response, established cryptographic session, green dashboard, or closed ticket is not sufficient proof. Intended state must be reconciled with applied state, negative tests, degraded behavior, raw evidence, recovery, and the resulting security or mission 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 authorized and prohibited behavior, inject realistic failures, measure convergence or recovery, and retain a tested rollback. Automation must stop safely when authority, freshness, identity, integrity, cryptographic state, or observed outcomes are ambiguous.

Troubleshooting locates the first divergence from the expected state machine or end-to-end transaction. Inspect every authority boundary and account for time, caching, eventual consistency, stale identity, incomplete collection, parser drift, asymmetric

paths, encapsulation, supply-chain state, side channels, partial failure, and missing evidence. Closure requires causal explanation, reproducibility, validated restoration, residual limitations, and prevention or monitoring actions.

Engineering practices

- Define authority, ownership, scope, dependencies, limits, and acceptance criteria for testing and evidence.
- Model expected state, trust boundaries, data flow, identity, and failure behavior before implementation.
- Validate intended configuration, active state, applied enforcement, raw evidence, and end-to-end outcome independently.
- Test normal, prohibited, degraded, failed, scale, recovery, and rollback conditions.
- Preserve policy, source, version, identity, telemetry, artifact, packet, log, time, and decision evidence.
- Automate through typed contracts, least privilege, deterministic gates, canaries, bounded authority, and reversible changes.

Failure patterns

- Treating testing and evidence as a product feature instead of an end-to-end engineering system.
- Relying on intended policy, rule code, or controller state without proving applied and operational behavior.
- Ignoring data quality, identity, time, cryptographic lifecycle, supplier state, 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 result without negative tests, rollback, and recurrence monitoring.

Validation and evidence

Method	Expected evidence
Examine	Requirements, architecture, identities, data, algorithms, policy, configuration, ownership, and dependencies for testing and evidence.
Observe	Active state, applied policy, raw events, telemetry, artifacts, cryptographic results, resources, and endpoint behavior.
Test	Expected behavior, prohibited behavior, malformed inputs, dependency loss, partial failure, scale, recovery, and rollback.
Measure	Availability, latency, detection quality, evidence completeness, cryptographic cost, risk, resource use, and mission outcome.
Reconcile	Intended state against observed security, identity, application, cloud, cryptographic, data, and evidence records.

8.4 Adoption and maturity gates

Adoption and maturity gates must be implemented as an observable engineering mechanism, not a product label or maturity claim. Separate standards, community references, production platforms, research, benchmarks, audits, interoperability, economics, and skills. The design should name authoritative inputs, identities, state transitions, dependencies, cryptographic or policy boundaries, limits, owners, telemetry, failure behavior, and acceptance criteria.

This guide traces the subject through the workload, TEE, evidence, verifier, appraisal, key release, protected computation, output, and relying-party decision chain. A configured integration, successful API response, established cryptographic session, green dashboard, or closed ticket is not sufficient proof. Intended state must be reconciled with applied state, negative tests, degraded behavior, raw evidence, recovery, and the resulting security or mission 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 authorized and prohibited behavior, inject realistic failures, measure convergence or recovery, and retain a tested rollback. Automation must stop safely when authority, freshness, identity, integrity, cryptographic state, or observed outcomes are ambiguous.

Troubleshooting locates the first divergence from the expected state machine or end-to-end transaction. Inspect every authority boundary and account for time, caching, eventual consistency, stale identity, incomplete collection, parser drift, asymmetric paths, encapsulation, supply-chain state, side channels, partial failure, and missing evidence. Closure requires causal explanation, reproducibility, validated restoration, residual limitations, and prevention or monitoring actions.

Engineering practices

Define authority, ownership, scope, dependencies, limits, and acceptance criteria for adoption and maturity gates.

Model expected state, trust boundaries, data flow, identity, and failure behavior before implementation.

Validate intended configuration, active state, applied enforcement, raw evidence, and end-to-end outcome independently.

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

Preserve policy, source, version, identity, telemetry, artifact, packet, log, time, and decision evidence.

Automate through typed contracts, least privilege, deterministic gates, canaries, bounded authority, and reversible changes.

Failure patterns

Treating adoption and maturity gates as a product feature instead of an end-to-end engineering system.

Relying on intended policy, rule code, or controller state without proving applied and operational behavior.

Ignoring data quality, identity, time, cryptographic lifecycle, supplier state, 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 result without negative tests, rollback, and recurrence monitoring.

Validation and evidence

Method	Expected evidence
Examine	Requirements, architecture, identities, data, algorithms, policy, configuration, ownership, and dependencies for adoption and maturity gates.
Observe	Active state, applied policy, raw events, telemetry, artifacts, cryptographic results, resources, and endpoint behavior.
Test	Expected behavior, prohibited behavior, malformed inputs, dependency loss, partial failure, scale, recovery, and rollback.
Measure	Availability, latency, detection quality, evidence completeness, cryptographic cost, risk, resource use, and mission outcome.
Reconcile	Intended state against observed security, identity, application, cloud, cryptographic, data, and evidence records.

Chapter 08 laboratory and review

Practical laboratory

Create an adoption decision for a sensitive AI or analytics workload comparing conventional cloud, TEE, FHE, ZKP, and hybrid designs.

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 verifiable confidential-computing fabric with post-quantum attestation, privacy-enhancing analytics, and governed AI operations.

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. Confidential Computing Consortium - Confidential Computing Consortium
Role: Confidential-computing definition, projects, white papers, and ecosystem resources.
Verified: 2026-07-26
Official source: https://confidentialcomputing.io/

02. Confidential Computing Consortium - Why Is Attestation Required for Confidential Computing?
Role: Industry explanation of hardware-based, attested TEEs and data-in-use protection.
Verified: 2026-07-26
Official source: https://confidentialcomputing.io/2023/04/06/why-is-attestation-required-for-confidential-computing/

03. IETF / RFC Editor - RFC 9334 - Remote ATtestation procedureS Architecture
Role: Architecture and terminology for attesters, verifiers, relying parties, evidence, and appraisal.
Verified: 2026-07-26
Official source: https://www.rfc-editor.org/info/rfc9334

04. IETF / RFC Editor - RFC 9711 - The Entity Attestation Token
Role: Proposed Standard for attestation-oriented claims in CWT or JWT form.
Verified: 2026-07-26
Official source: https://www.rfc-editor.org/info/rfc9711

05. IETF / RFC Editor - RFC 9783 - Arm's Platform Security Architecture Attestation Token
Role: Platform-specific attestation token mapped to the RATS architecture.
Verified: 2026-07-26
Official source: https://www.rfc-editor.org/info/rfc9783

06. NIST - Privacy-Enhancing Cryptography Project
Role: NIST research and standardization context for FHE, ZKP, MPC, PSI, and related tools.
Verified: 2026-07-26
Official source: https://csrc.nist.gov/Projects/pec

07. NIST - Privacy-Enhancing Cryptography - Fully Homomorphic Encryption
Role: Current NIST FHE project and use-case context.
Verified: 2026-07-26
Official source: https://csrc.nist.gov/projects/pec/fhe

08. NIST - Privacy-Enhancing Cryptography - Zero-Knowledge Proofs
Role: NIST ZKP project and collaboration context.
Verified: 2026-07-26
Official source: https://csrc.nist.gov/Projects/pec/zkproof

09. HomomorphicEncryption.org - Homomorphic Encryption Security Guidelines
Role: Community-developed FHE security guidance; not represented as a formal government standard.
Verified: 2026-07-26
Official source: https://homomorphicencryption.org/security-guidelines/

10. ZKProof - ZKProof Community Reference
Role: Living community reference for secure and interoperable ZKP development and deployment.
Verified: 2026-07-26
Official source: https://docs.zkproof.org/reference

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-EMT; MYTHOS-AI; MYTHOS-CLD
Publication class	Field Guide / Canonical Living Overview Guide
Skill levels	L1 Foundational; L2 Practitioner; L3 Advanced; L4 Frontier
Tags	confidential computing, TEE, remote attestation, RATS, EAT, FHE, zero knowledge proofs, privacy enhancing cryptography, confidential AI, advanced assurance
Canonical route	/library/field-guides/mythos-fg-sec-0018/

Living publication

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