International Journal of Modern Computer Science and IT Innovations (IJMCSIT)
eISSN: 3087-4289
Volume03 Issue10 | Page No: -18-41

Received: 24 Sep 2026, Revised: 05 Oct 2026, Accepted: 16 Oct 2026, Published: 22 Oct 2026

DOI: https://doi.org/10.55640/ijmcsit-v03i10-03

Confidential Computing Pipelines for Kubernetes Observability Data: An Attested Architecture for Logs, Metrics, Traces, and Audit Events

Soham Dasgupta
Independent Researcher, USA

ABSTRACT: Kubernetes observability pipelines collect logs, metrics, traces, and audit events that are essential for reliability engineering, incident response, compliance, and capacity planning. However, these same signals frequently contain sensitive application payloads, tenant identifiers, access tokens, internal topology, operational metadata, and personally identifiable information. Conventional protections--transport encryption, storage encryption, role-based access control, and backend retention policies--do not protect telemetry while it is being parsed, enriched, sampled, correlated, redacted, routed, or queried. This creates a security gap: the observability pipeline itself becomes a privileged data-processing system.

This paper proposes ConfObs-TEE, a confidential computing architecture for Kubernetes observability data. This design-science study presents a reference architecture and a prospective prototype-evaluation plan; it does not report measured implementation results. The architecture places the highest-risk telemetry operations inside hardware-backed, remotely attested Trusted Execution Environments (TEEs), while preserving compatibility with OpenTelemetry Collector patterns and Kubernetes-native deployment models. ConfObs-TEE uses attested collector agents, confidential gateway collectors, policy-bound key release, signed pipeline manifests, field-level tokenization, envelope encryption, and verifiable lineage metadata to reduce trust in cluster nodes, cloud administrators, telemetry operators, and observability backends. The design is grounded in current cloud-native mechanisms, including OpenTelemetry Collector receivers, processors, and exporters; Kubernetes metadata enrichment; Confidential Containers; Key Broker Service-style attestation; and the IETF RATS attestation model. The paper contributes a threat model for observability data in Kubernetes, a reference architecture, a policy and key-management design, a security analysis, and an evaluation methodology for future prototype validation. The central design conclusion is that confidential computing should not be applied indiscriminately to the entire observability stack, but selectively to the raw-data, enrichment, correlation, redaction, and key-handling stages where plaintext exposure and operator privilege are most consequential.

KEYWORDS: confidential computing, Kubernetes, observability, OpenTelemetry, trusted execution environment, telemetry privacy, remote attestation, logs, traces, metrics, audit events.

1. Introduction

Kubernetes has become a dominant substrate for deploying microservices, data platforms, and cloud-native applications. Operating such systems requires continuous observability: cluster administrators and site reliability engineers collect metrics, logs, traces, events, and audit records to infer system health, diagnose failures, detect abuse, and support compliance workflows. Kubernetes documentation describes observability as the collection and analysis of metrics, logs, and traces to understand the internal state, performance, and health of a cluster [3]. Kubernetes auditing additionally produces chronological security-relevant records of API-server activity, including who performed an action, when it occurred, where it originated, and what resource it affected [4].

The observability pipeline is therefore not merely an operational utility. It is a high-value data pipeline. Logs

may include request bodies, email addresses, customer identifiers, authorization headers, stack traces, database query fragments, or error dumps. Traces may include URLs, headers, baggage values, database statements, and service dependency graphs. Metrics may reveal tenant behavior, workload scale, business volume, or security-sensitive labels. Audit events may reveal administrative actions, service-account behavior, namespace structure, and security-policy changes. Empirical work on software logs reports that sensitive information in logs remains a privacy concern [26], and the OpenTelemetry project explicitly states that implementers--not OpenTelemetry itself--are responsible for protecting sensitive telemetry and complying with applicable data-handling requirements [9].

Modern observability stacks typically protect telemetry in transit with TLS and at rest with backend encryption. These controls are necessary but incomplete. Telemetry must be decrypted and processed by collectors, agents, gateways, stream processors, search engines, dashboards, and alerting systems. In Kubernetes, collectors often run as DaemonSets, sidecars, or gateway Deployments, and they commonly enrich telemetry with Kubernetes metadata before exporting it to one or more backends. The OpenTelemetry Collector is explicitly designed as a vendor-agnostic component that receives, processes, and exports telemetry through receivers, processors, exporters, and connectors [5], [6].

This design creates a trust gap. Cluster nodes, host operating systems, privileged DaemonSets, Kubernetes administrators, cloud operators, and observability-platform administrators may all be able to access plaintext telemetry at some stage. Encryption at rest and in transit does not protect data while processors parse, redact, enrich, sample, aggregate, or route it. Confidential computing addresses this missing protection by safeguarding data in use through hardware-based, attested TEEs. The Confidential Computing Consortium defines confidential computing as protecting data in use by performing computation in a hardware-based, attested TEE [1], [2].

Kubernetes observability is therefore a focused target for selective confidential computing. The challenge is not simply to "run the monitoring stack inside an enclave." Observability systems are high-throughput, latency-sensitive, highly integrated, and operationally critical. They require access to Kubernetes metadata, workload identity, application events, logs on node filesystems, kubelet metrics, audit streams, external storage systems, and incident-response tools. A naive confidential-computing design may either protect too little--leaving raw telemetry exposed before enclave entry--or protect too much--creating cost, complexity, and availability risks. As §2.5 shows, secure logging, private telemetry, confidential analytics, and confidential containers each provide relevant mechanisms, but they address different stages and utility goals.

To address this design space, we propose ConfObs-TEE, a selective confidential computing architecture for Kubernetes observability data. ConfObs-TEE protects the stages where plaintext telemetry and secret material are most exposed: ingestion of raw telemetry, sensitive-field classification, Kubernetes metadata enrichment, tokenization, redaction, sampling, correlation, export-credential use, and optional break-glass query decryption. The design uses remote attestation to bind keys and credentials to measured collector images, configurations, policies, and runtime environments. It allows sanitized telemetry to flow to conventional observability backends while raw telemetry is encrypted, minimized, or held only in confidential execution contexts.

The contributions of this paper are:

- A Kubernetes-specific threat model for observability pipelines.
- A reference architecture for confidential telemetry ingestion, enrichment, transformation, and export.
- A policy-bound key-release model based on remote attestation and signed pipeline manifests.
- A data-governance model for classifying, redacting, tokenizing, and encrypting logs, metrics, traces, and audit events.
- A security analysis identifying protected assets, mitigated threats, residual risks, and deployment trade-offs.
- A reproducible, prospective evaluation methodology for measuring confidentiality, integrity, throughput, latency, resource overhead, and operational utility once a prototype exists.

This manuscript is written as a design-science paper. Accordingly, it reports no prototype benchmark results; §9 defines a rigorous validation plan that can be executed using Kubernetes, OpenTelemetry Collector, Confidential Containers, and a selected hardware-backed TEE platform.

2 Background and Motivation

2.1 Kubernetes observability data

Kubernetes observability data can be grouped into four major signal classes.

Metrics are numeric measurements such as CPU usage, memory usage, restart counts, request latency, queue depth, and error rates. Metrics are usually aggregated and lower in payload sensitivity than logs or traces, but labels can still reveal sensitive information such as tenant IDs, customer IDs, namespace names, deployment topology, or workload scale.

Logs are semi-structured or unstructured records emitted by applications, system components, containers, and nodes. They are often the richest and riskiest telemetry signal because developers may log request bodies, exception payloads, stack traces, configuration values, tokens, or user-provided data.

Traces represent request paths across services. Distributed traces may reveal internal service graphs, endpoint names, database calls, message-queue topics, error paths, and identity context.

Audit events document Kubernetes API activity and are critical for security forensics. Kubernetes audit logs are generated by the API server and record security-relevant activity from users, applications, and the control plane [4].

The operational value of these signals increases when they are correlated. OpenTelemetry provides a common framework for instrumenting applications and collecting telemetry. The Collector receives telemetry from sources, applies processors, and exports data to one or more backends. In Kubernetes, important Collector components include Kubernetes metadata enrichment, kubelet statistics receivers, and file log receivers [5], [8], [14].

2.2 Observability pipelines as sensitive data processors

A common Kubernetes observability deployment contains node-local agents and one or more gateway collectors. The node-local agent may collect container logs from node filesystems, scrape kubelet or cAdvisor metrics, receive OTLP traffic from applications, and enrich records with pod or namespace metadata. A gateway collector may batch, sample, route, normalize, correlate, and export data to backends. Official OpenTelemetry documentation describes Collector deployment on Kubernetes using DaemonSet and gateway-style components, and the Collector's configuration model is based on receivers, processors, exporters, and connectors [6], [7].

This architecture centralizes sensitive material. The collector may hold:

- Raw logs and trace spans;
- Kubernetes metadata and labels;
- Backend credentials;
- Tenant routing rules;
- Sampling decisions;
- Redaction policies;
- Temporary buffers;
- TLS private keys;
- Audit records; and
- Error queues and retry buffers.

If the collector, host, node administrator, or backend operator is compromised, telemetry becomes an exfiltration path. This risk is amplified because observability systems are often designed to maximize visibility, not minimize data.

2.3 Confidential computing and remote attestation

Confidential computing protects data while it is in use. A hardware-backed TEE is intended to isolate code and data from software outside its declared trust boundary. The exact protection against a host, hypervisor, cloud administrator, physical attacker, or side channel depends on the selected TEE platform, firmware, configuration, and attestation policy; the architecture therefore does not treat SGX, SEV-SNP, TDX, CCA, or confidential virtual machines as interchangeable [15]. The CCC frames the common objective around hardware-based, attested TEEs [1], [2].

Remote attestation is the mechanism by which a relying party decides whether a remote workload is running in an expected environment. The IETF RATS architecture

defines roles and messages for attestation, including evidence, endorsements, reference values, appraisal policies, attestation results, attesters, verifiers, and relying parties [10].

In cloud-native systems, Confidential Containers extend confidential computing to Kubernetes workloads. Confidential Containers is a CNCF Sandbox project that uses TEEs to protect containers and data, and its trust model maps pods to confidential virtual machines in which the guest does not trust the host [12], [13]. The platform-specific confidential runtime and its documented threat model determine which host and hypervisor capabilities are actually excluded from the trust boundary.

Confidential Containers also uses attestation and secret-release patterns. Its Key Broker Service architecture validates evidence through an attestation service and releases resources only after policy evaluation; the KBS protocol is designed to bind a secure channel to fresh attestation evidence [11].

2.4 Why observability needs confidential computing

Telemetry processing has three properties that make it suitable for selective confidential computing.

First, telemetry is security-sensitive even when it is not business data. Logs and traces can contain credentials, PII, internal endpoints, configuration mistakes, access patterns, and dependency graphs. OpenTelemetry's security guidance emphasizes that the implementer is responsible for protecting sensitive telemetry and that attribute, filter, redaction, and transform processors can help remove or modify sensitive data [9].

Second, telemetry requires active processing. It is not enough to store encrypted data. Collectors must parse, tag, transform, sample, correlate, and route data. These operations require plaintext access unless confidential computing or advanced cryptography is used.

Third, observability crosses administrative boundaries. In many organizations, application teams, platform teams, cloud providers, security teams, and external observability vendors all interact with telemetry. Confidential computing can reduce the number of parties that must be trusted with plaintext telemetry.

2.5 Related work and positioning

Secure-logging research established forward integrity, tamper evidence, and protected log storage before cloud TEEs were widely deployed. Schneier and Kelsey protect the integrity of audit logs even after a logging host is compromised [27], while Ma and Tsudik develop a secure-logging construction without requiring an online trusted third party [28]. SGX-Log and EmLog place security-critical logging functions inside TEEs [29], [30]. These systems are close precedents for protecting log creation or storage, but they do not address a Kubernetes-native, multi-signal pipeline that enriches, samples, tokenizes, routes, and selectively discloses telemetry.

Attestation and confidential-computing systems provide complementary precedents. Keylime demonstrates scalable cloud trust bootstrapping and attestation-conditioned provisioning [31]. VC3 and Opaque protect distributed analytics, while SCONE supports secure Linux containers and StealthDB protects database query processing [32]–[35]. Their protected computation generally occurs at the analytics, container-runtime, or database layer. ConfObs-TEE instead moves the boundary toward telemetry ingestion and enrichment, then deliberately permits sanitized derivatives to use conventional backends.

Privacy-preserving telemetry systems optimize different disclosure goals. Prochlo combines trusted shuffling, data minimization, and differential privacy for crowd analytics [36]. Prio securely aggregates statistics without exposing individual contributions [37], and RAPPOR applies randomized response to telemetry collection [38]. Network-trace anonymization and privacy-preserving log filtering study the privacy–utility trade-off for operational data [39], [40]. These approaches can provide stronger privacy for aggregate reporting, but they generally do not preserve the same record-level debugging workflow or controlled raw-evidence path. ConfObs-TEE does not claim differential privacy: metadata, timing, volume, and correlation leakage remain residual risks.

Table 1. Positioning against secure logging, confidential analytics, and private telemetry.

Work family	Protected stage / signal	Mechanism and trust boundary	Position relative to ConfObs-TEE
Secure audit logging [27], [28]	Log creation and stored audit records	Cryptographic forward integrity and tamper evidence	Strong integrity precedent; does not cover Kubernetes enrichment, multi-signal routing, or selective backend disclosure.
TEE-based logging [29], [30]	System-log capture and protected logging	Logging logic and keys inside SGX or another TEE	Most direct confidential-logging precedent; narrower signal and deployment scope.
Keylime [31]	Cloud workload bootstrap	Remote attestation conditions trust and provisioning	Precedent for scalable appraisal and key release; not an observability data plane.
VC3, Opaque [32], [33]	Distributed analytics	Enclaves protect analytics computation; Opaque also addresses access patterns	Protects later analytics stages rather than collection, Kubernetes metadata enrichment, and sanitization.
SCONE, StealthDB [34], [35]	Container runtime and database queries	Confidential containers or enclave-protected database execution	Runtime and query precedents; no multi-signal observability governance model.
Prochlo, Prio, RAPPOR [36]–[38]	Crowd telemetry and aggregate statistics	Trusted shuffling, secure aggregation, or local differential privacy	Stronger aggregate privacy; generally sacrifices the same record-level debugging and controlled raw-evidence path.
Trace anonymization and DeLog [39], [40]	Network traces and filtered logs	Anonymization or privacy-preserving filtering	Directly studies privacy–utility trade-offs, but not attestation-bound Kubernetes processing.
ConfObs-TEE (this paper)	Logs, metrics, traces, and audit events from ingress through export	Attested collectors and gateway, policy-bound capabilities, sanitized backend, encrypted raw vault	Integrates prior mechanisms into a Kubernetes-native design; implementation cost and benefit remain prospective.

Table 1 positions the proposal as a composition of established mechanisms rather than a claim that each mechanism is individually new. Its distinctive design scope is the combination of Kubernetes-native multi-signal collection, attestation-bound pipeline identity, confidential enrichment and transformation, a sanitized-backend boundary, and a separately governed raw-data path. Whether that combination is practical and beneficial remains an empirical question for the prospective evaluation in §9.

3 Problem Statement and Requirements

3.1 Problem statement

Given a Kubernetes cluster that emits logs, metrics, traces, events, and audit records, design an observability pipeline that allows operators to monitor, debug, and investigate systems while minimizing plaintext telemetry exposure to untrusted hosts, Kubernetes administrators, cloud operators, external observability platforms, and compromised pipeline components.

The problem is difficult because telemetry must remain useful. Over-redaction harms incident response. Excessive encryption prevents search, correlation, and alerting. Full-stack confidential execution increases cost and operational complexity. Therefore, the design must selectively protect the most sensitive processing stages while maintaining cloud-native compatibility.

3.2 Design requirements

Table 2. ConfObs-TEE design requirements.

ID	Requirement	Rationale
R1	Protect raw telemetry in use	Logs, traces, and audit records may contain secrets or regulated data during processing.
R2	Bind secrets to attested code and configuration	Backend credentials, tokenizer keys, and decryption capabilities must not be released to unmeasured collectors.
R3	Preserve Kubernetes-native deployment	The pipeline should work with DaemonSets, Deployments, sidecars, namespaces, service accounts, and OpenTelemetry Collector conventions.
R4	Support data minimization	Sensitive fields should be dropped, masked, tokenized, encrypted, or routed according to policy.
R5	Preserve operational utility	Sanitized telemetry should remain searchable, aggregatable, and useful for debugging.
R6	Support multi-tenancy	Tenant, namespace, and workload boundaries must map to separate keys and policies.
R7	Provide verifiable lineage	Operators should be able to verify which collector, configuration, manifest, and policy processed a telemetry record.
R8	Fail safely	Attestation failure should prevent confidential key and normal export-credential release, not silently downgrade confidentiality.

ID	Requirement	Rationale
R9	Expose residual risk	Side channels, malicious applications, telemetry poisoning, metadata leakage, and availability attacks must be documented rather than hidden.

The requirements in Table 2 make the protection boundary and its limits explicit. In particular, R8 prevents an orchestration identity or unattested collector from obtaining secret-bearing capabilities, while R9 requires the design to report residual exposure rather than treating the TEE as a complete security boundary.

4 Threat Model

4.1 Assets

The protected assets include:

- Raw logs, traces, metrics, audit events, and Kubernetes events;
- Sensitive attributes, labels, headers, URLs, request bodies, stack traces, and exception payloads;
- Kubernetes metadata such as namespace, pod UID, node name, deployment, labels, annotations, and service account;
- Telemetry routing rules and tenant mappings;
- Redaction, tokenization, and sampling policies;
- Backend export credentials;
- Encryption keys, HMAC keys, and signing keys;
- Intermediate buffers, queues, retry files, and crash dumps; and
- Lineage records and audit trails.

4.2 Adversaries

The architecture considers the following adversaries.

A1: Curious or compromised host operator. The adversary controls the Kubernetes worker node, host operating system, container runtime, or hypervisor and attempts to inspect collector memory, logs, files, or network connections. Protection from a particular host or hypervisor capability is claimed only when it is excluded by the selected TEE platform's threat model.

A2: Malicious cloud or infrastructure administrator. The adversary has privileged infrastructure access but should not learn tenant telemetry in plaintext within the protection scope of the selected TEE.

A3: Compromised non-confidential collector. The adversary compromises a conventional collector or gateway and attempts to access raw telemetry or export credentials.

A4: Observability backend insider. The adversary can query or administer the backend but should only see sanitized telemetry unless approved break-glass decryption occurs.

A5: Network attacker. The adversary intercepts or modifies telemetry in transit.

A6: Malicious workload. A workload intentionally emits misleading, excessive, or secret-containing telemetry.

A7: Kubernetes administrator with scheduling control. The adversary can create or mutate Kubernetes resources but should not cause secrets to be released to an unapproved collector image or configuration.

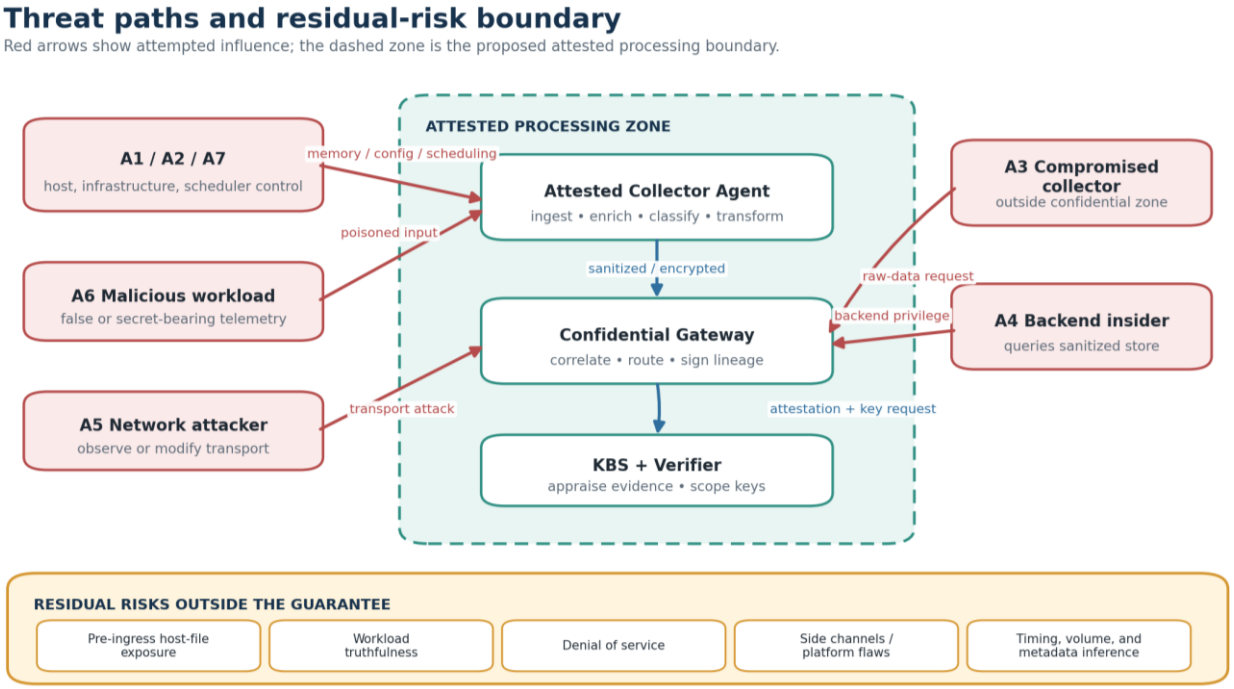


Figure 1. Threat paths, attack surfaces, and residual risks outside the proposed guarantee.

Figure 1 maps each adversary to the first component or boundary it can influence. It also separates threats the design attempts to mitigate from residual risks such as pre-ingress host-file exposure, workload truthfulness, denial of service, and side channels. The diagram therefore represents a conditional security argument, not a claim that attestation makes all infrastructure trustworthy.

4.3 Assumptions

The design assumes that the selected TEE platform, firmware, microcode, attestation service, and cryptographic libraries are within the trusted computing base defined for that deployment. It also assumes that reference measurements for approved collector images, configurations, manifests, and policies are maintained correctly. The Kubernetes control plane may be useful for orchestration, but it is not trusted to authorize access to sensitive telemetry keys by itself. This follows the broader confidential-Kubernetes observation that managed control planes may be outside the workload owner's attestation boundary [12].

Because TEE guarantees differ, a deployment must identify the concrete platform and document its protection scope. For example, process-enclave and confidential-VM designs have different trusted computing bases, I/O paths, memory-encryption semantics, device assumptions, and attestation evidence. The abstract architecture in this paper specifies the properties ConfObs-TEE needs; it does not assert that every platform supplies them identically.

4.4 Out-of-scope threats

ConfObs-TEE does not fully prevent:

- Application code from intentionally logging secrets;
- Denial-of-service attacks against collectors, nodes, networks, or KBS services;
- All side-channel attacks against TEEs;
- Compromised TEE hardware, firmware, or attestation roots;
- Inference from sanitized metadata, timing, volume, or label cardinality; or
- Data exposure before telemetry enters an attested component in node-file log collection mode.

These exclusions are important. TEE systems have a documented history of side-channel and fault-injection concerns. Surveys of TEE insecurity and attacks such as Plundervolt show that privileged or microarchitectural adversaries can undermine particular TEE guarantees under particular conditions [15], [16]. FABRICKED further illustrates that confidential computing depends on correct platform initialization, firmware behavior, and attestation integrity, not simply on encryption being advertised by hardware [17], [18].

5 Proposed Architecture: ConfObs-TEE

5.1 Architectural overview

ConfObs-TEE divides the observability stack into confidential and non-confidential zones.

The **confidential zone** contains raw-telemetry ingestion, sensitive-field classification, Kubernetes metadata enrichment, redaction, tokenization, envelope encryption, sampling decisions that require sensitive context, export-credential use, and optional raw-data query decryption.

The **non-confidential zone** contains sanitized telemetry storage, dashboards, alerting, long-term metric aggregation, public metadata indexing, and low-sensitivity analytics.

The core design principle is:

> **Raw telemetry and secret-bearing operations occur only inside measured, attested execution environments; sanitized and policy-approved derivatives may leave the TEE.**

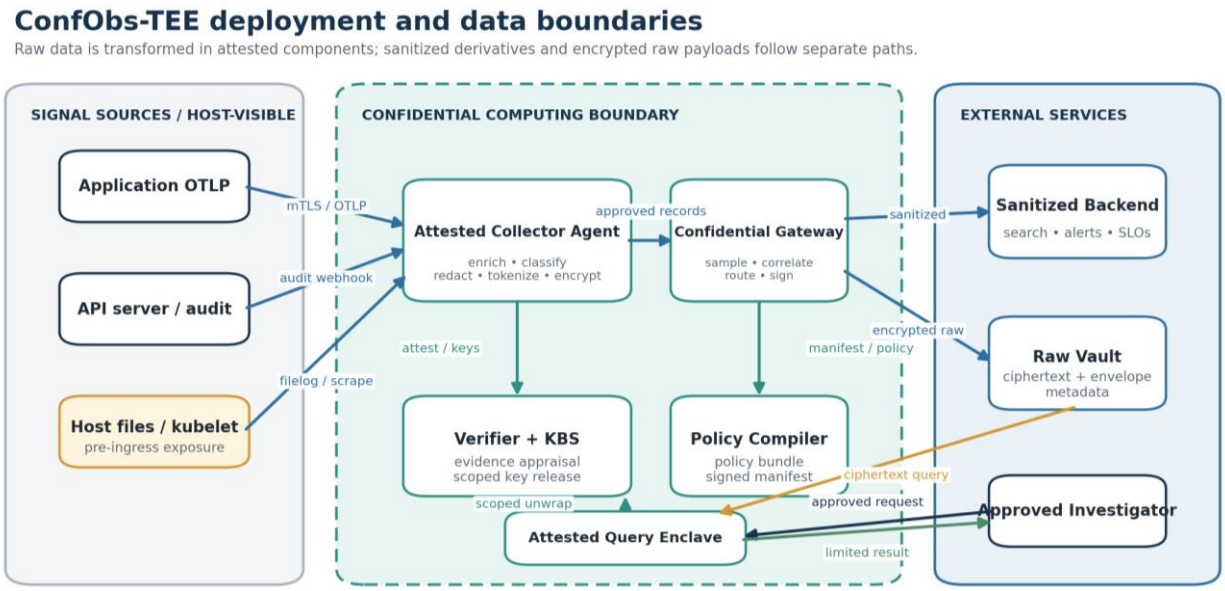


Figure 2. ConfObs-TEE deployment, data flows, and confidential-computing trust boundaries.

Figure 2 replaces the textual pipeline sketch with an explicit deployment and trust-boundary view. Protection begins only when a signal crosses into an attested component: direct OTLP and confidential-sidecar paths avoid an additional host-visible file stage, whereas node-file collection retains a pre-ingress exposure. Sanitized derivatives can be queried in conventional backends; raw ciphertext follows a separate storage and access path so backend administration does not imply raw-data access.

5.2 Components

5.2.1 Attested Collector Agent

The Attested Collector Agent is a Kubernetes-deployed collector that runs inside a TEE-backed pod or confidential VM. It can be deployed as:

- A sidecar for highly sensitive workloads;
- A node-local DaemonSet for cluster-wide collection;
- A namespace-local collector for tenant isolation; or
- A dedicated audit-event collector for Kubernetes API-server audit streams.

The agent receives telemetry through OpenTelemetry-compatible receivers, applies processors, and exports sanitized or encrypted data. This aligns with OpenTelemetry's collector model, in which receivers collect data, processors transform or filter it, and exporters send it onward [5], [6].

The agent is measured at startup. Its image digest, configuration digest, policy-bundle digest, signed-manifest digest, and optional plugin digests are included in reference values used by the verifier or otherwise bound into the attested session. The precise binding mechanism is platform-dependent and must be implemented and tested for the selected runtime. Secrets are not mounted as ordinary Kubernetes Secrets. Instead, the agent must attest before receiving keys or backend credentials.

5.2.2 Confidential Gateway Collector

The Confidential Gateway Collector aggregates traffic from multiple agents. It performs operations that require a wider view of traces or cross-service flows, such as:

- Tail-based sampling;
- Tenant-level rate limiting;
- Multi-signal correlation;
- Deduplication;
- Export routing;
- Raw-telemetry vault encryption; and
- Lineage signing.

Gateway collectors can be horizontally scaled. Each instance attests independently and receives only the keys needed for its tenant and signal scope.

5.2.3 Attestation and Key Broker Service

The Attestation and Key Broker Service verifies evidence, checks reference values, evaluates policy, and releases secrets only to approved collector instances. This mirrors the Confidential Containers KBS pattern, where a KBS uses an attestation service to validate evidence and releases resources after policy evaluation [11]. Keylime provides a complementary precedent for scalable attestation and trust bootstrapping [31].

The KBS may release:

- Short-lived mTLS certificates;
- Exporter credentials;
- Scoped key-encryption or data-encryption material for envelope encryption;
- HMAC/tokenization keys;
- Raw-telemetry vault capabilities; and
- Policy-specific decryption capabilities.

The KBS does not release broad, long-lived cluster credentials. Key release is scoped by workload identity, namespace, collector measurement, signed-manifest and policy versions, tenant or signal scope, and time epoch.

5.2.4 Policy Compiler and signed pipeline manifest

The Policy Compiler transforms human-readable telemetry-governance rules into machine-enforced policy bundles. Policies define:

- Which attributes are public, confidential, restricted, or prohibited;
- Which fields are dropped, masked, tokenized, encrypted, or preserved;
- Which backends may receive each signal;
- Retention classes;
- Namespace and tenant routing;
- Break-glass access rules; and
- Allowed collector images and configurations.

OPA/Rego is a natural fit for expressing portable policy-as-code. OPA is a general-purpose policy engine used across applications, Kubernetes, CI/CD, API gateways, and other systems [19].

The compiler also emits a **signed pipeline manifest**: a versioned, content-addressed artifact that lists the approved collector image, configuration and policy digests, optional plugin digests, permitted receivers/processors/exporters, destination classes, tenant or namespace scope, and credential classes. A release authority signs the manifest using the organization's software-supply-chain signing process. The verifier treats the manifest digest as a reference input and the collector exposes that digest in its attested session. The manifest is distinct from per-record lineage metadata: it defines the approved pipeline identity, while lineage records what processed a particular batch. This is an architectural artifact whose concrete schema and signing workflow must be validated in a prototype.

5.2.5 Sanitized Observability Backends

Conventional backends receive only sanitized telemetry or encrypted payloads. For example:

- Metrics may retain service-level labels but drop tenant-specific high-cardinality labels;
- Traces may retain span structure but tokenize user IDs and remove authorization headers;
- Logs may retain severity, service, namespace, error class, and tokenized correlation IDs while encrypting raw message bodies; and
- Audit events may retain action metadata while encrypting request bodies for sensitive resources.

5.2.6 Break-glass Query Enclave

Some incidents require access to raw telemetry. ConfObs-TEE supports this through a separate attested query enclave. Rather than granting backend administrators

direct decryption keys, the system requires an attested query environment, policy approval, auditable justification, time-limited credentials, and least-privilege decryption.

6 Dataflow Design

6.1 Startup and attestation

At startup, each confidential collector performs the following sequence:

1. Boot inside a TEE-backed pod or confidential VM.
2. Generate an ephemeral key pair inside the TEE.
3. Request attestation evidence from the platform.
4. Bind the ephemeral public key, collector-image digest, configuration digest, policy digest, signed-manifest digest, and nonce into the evidence or attested session using a mechanism supported by the selected platform.
5. Send the evidence to the attestation service and KBS.
6. The verifier checks platform endorsements, measurements, freshness, reference values, TCB status, and appraisal policy.
7. If verification succeeds, the KBS releases short-lived, scope-limited secrets over an attestation-bound secure channel.
8. If verification fails, the collector enters a non-exporting quarantine state or fails closed.

This follows the RATS principle that an attester produces evidence so a relying party can decide whether to trust it [10].

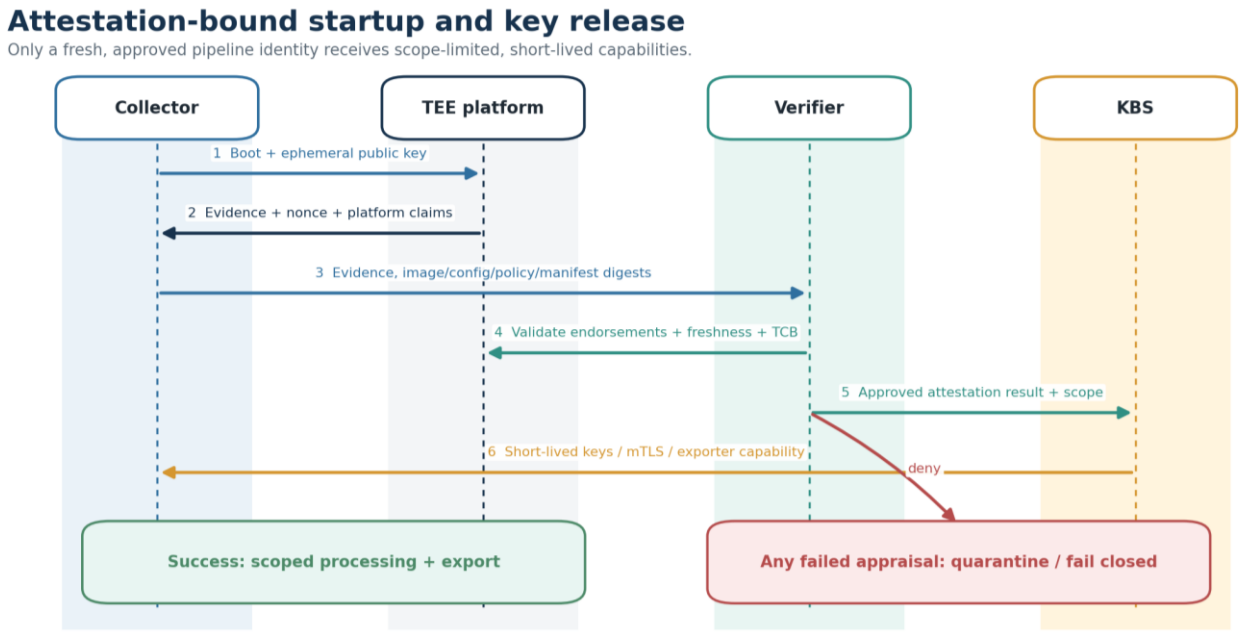


Figure 3. Attestation-bound collector startup and scope-limited key release.

Figure 3 makes the authorization dependency explicit. Kubernetes scheduling and workload identity can select a pod, but neither alone authorizes keys. Secret release depends jointly on fresh platform evidence, approved code and pipeline configuration, current policy and manifest digests, and the requested scope; any failed appraisal ends in quarantine or fail-closed behavior rather than plaintext export.

A policy sketch is shown below. It is illustrative rather than an implemented policy language:

```
release(secret_request)                                    allowed                                    when:
  tee.platform                                    in                                    approved_platforms
  and                    tee.tcb_status                    ==                    "up_to_date"
  and                    collector.image_digest            in                    approved_images
  and                    collector.config_digest           in                    approved_configs
  and                    collector.policy_digest           ==                    active_policy_digest
  and                    collector.manifest_digest         in                    approved_manifests
  and                    collector.namespace               in                    allowed_namespaces
  and                    secret_request.scope              subset_of             collector.allowed_scope
  and                    nonce                             is                    fresh
  and time < credential_expiry
```

6.2 Telemetry ingestion modes

ConfObs-TEE supports three ingestion modes.

Mode A: Direct confidential OTLP ingestion. Applications send telemetry directly to the Attested Collector Agent over mTLS. This mode provides the strongest confidentiality boundary because telemetry enters the confidential collector without first being written to host-visible log files.

Mode B: Confidential sidecar collection. A collector sidecar runs in the same confidential pod as the application. This mode is appropriate for highly sensitive workloads and can minimize exposure between application emission and telemetry processing.

Mode C: Node-file and kubelet collection. A node-local confidential agent reads container logs, kubelet metrics, and node-level signals. This mode is compatible with conventional Kubernetes logging patterns, but it has a capture-side gap: logs written to host-visible files may be exposed before the confidential collector reads them. Platform support for safely connecting a confidential pod or VM to host data sources must also be validated for the chosen runtime. For workloads with strict confidentiality requirements, direct OTLP or confidential-sidecar ingestion is preferred.

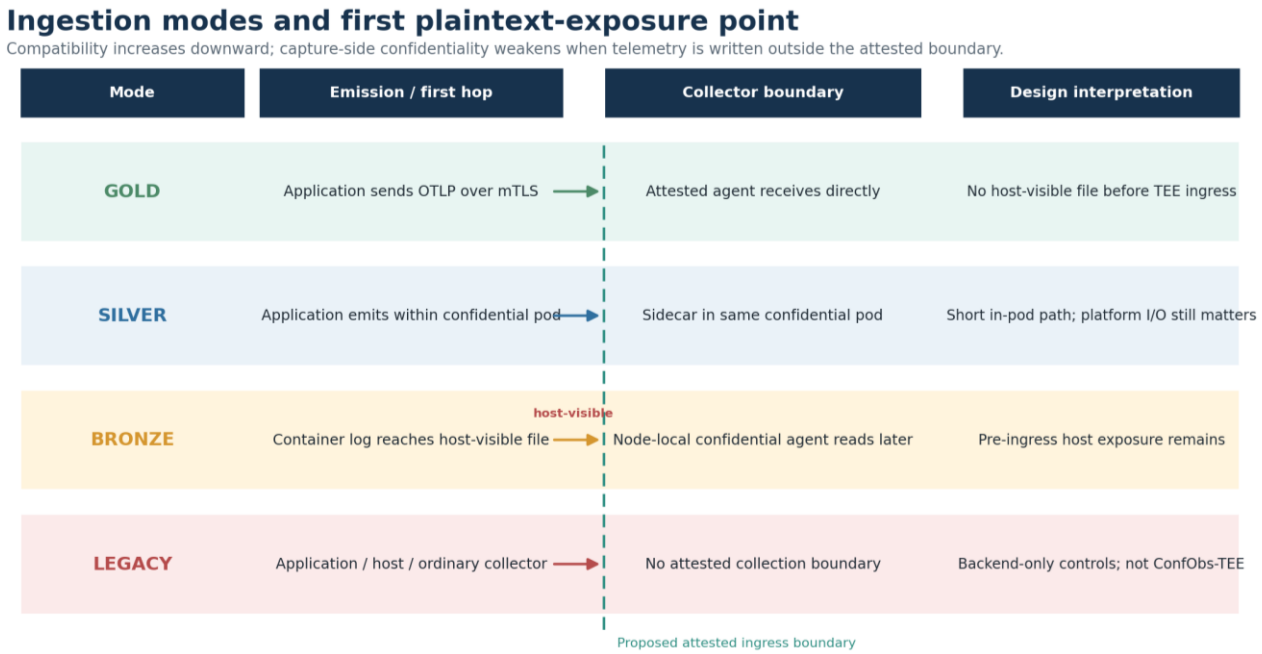


Figure 4. Ingestion modes and the first plaintext-exposure point relative to attested ingress.

Figure 4 compares compatibility with capture-side exposure. Direct OTLP and a same-confidential-pod sidecar cross into the attested boundary before host-visible persistence; node-file collection enters after that persistence point. The figure therefore supports the Gold/Silver/Bronze/Legacy classification in §10.2 without claiming that Bronze prevents pre-collection host access.

6.3 Enrichment

The Kubernetes attributes processor discovers pods, extracts metadata, and adds Kubernetes metadata to spans, metrics, and logs [14]. In ConfObs-TEE, metadata enrichment runs inside the confidential collector when the metadata itself is sensitive. The KBS policy constrains which Kubernetes API permissions are available to a collector. For example, a tenant collector may enrich only pods in its namespace, while a platform collector may enrich cluster-level telemetry but cannot access raw application logs from restricted namespaces.

6.4 Classification and transformation

ConfObs-TEE classifies fields into four categories:

Table 3. Default field-classification actions.

Class	Examples	Default action
Public operational	service name, severity, status code, coarse region	Preserve
Internal operational	namespace, pod UID, node name, deployment version	Preserve or hash according to tenant policy
Confidential	user ID, account ID, email, trace baggage, internal URL	Tokenize or encrypt
Restricted	authorization header, access token, secret value, sensitive request body	Drop, mask, or encrypt

OpenTelemetry processors can remove, modify, filter, redact, or transform sensitive telemetry [9]. ConfObs-TEE uses these capabilities inside a TEE and adds key-bound tokenization. This distinction matters: redaction outside a TEE still exposes sensitive data to the processor, while redaction inside a TEE reduces exposure to actors outside the selected platform's confidential-computing boundary.

6.5 Tokenization

Tokenization preserves operational utility while reducing plaintext exposure. For example:

```
user.email = "alice@example.com"
user.email_token = HMAC-SHA-256(k_tenant_epoch,
canonicalize("alice@example.com"))
user.email = "[redacted]"
```

Tokenization supports correlation without revealing the original value. Per-tenant, per-epoch keys limit blast radius. Rotation policies control whether tokens remain stable across days, incidents, or retention windows. A prototype must specify canonicalization, domain separation, key lifecycle, collision handling, and which fields are safe to tokenize rather than drop.

6.6 Raw telemetry vault and envelope encryption

For high-severity incidents, raw telemetry may be needed. ConfObs-TEE supports a raw-telemetry vault:

1. The raw record is encrypted inside the confidential collector.

2. The encrypted payload is exported to object storage or a log backend.

3. Indexable sanitized fields are exported separately.

4. Decryption requires an attested query enclave, policy approval, and audit logging.

Envelope encryption separates payload encryption from long-lived key authority. For each tenant, signal class, and policy-defined time epoch, the collector uses a data-encryption key (DEK) inside the TEE to protect the raw payload with an approved authenticated-encryption scheme. Tenant and lineage identifiers, policy and manifest digests, signal class, and epoch can be bound as authenticated associated data. The DEK is then wrapped by a scoped key-encryption key (KEK) or an equivalent KBS-protected capability. The stored envelope includes a versioned key identifier, nonce, wrapped DEK, ciphertext, and authentication tag; it does not expose the unwrapped DEK. Rotation creates a new epoch or key version, and revocation prevents new unwrap operations without claiming retroactive deletion of already released plaintext. The exact cryptographic suite, record format, and rewrapping procedure remain prototype decisions and must be externally reviewed.

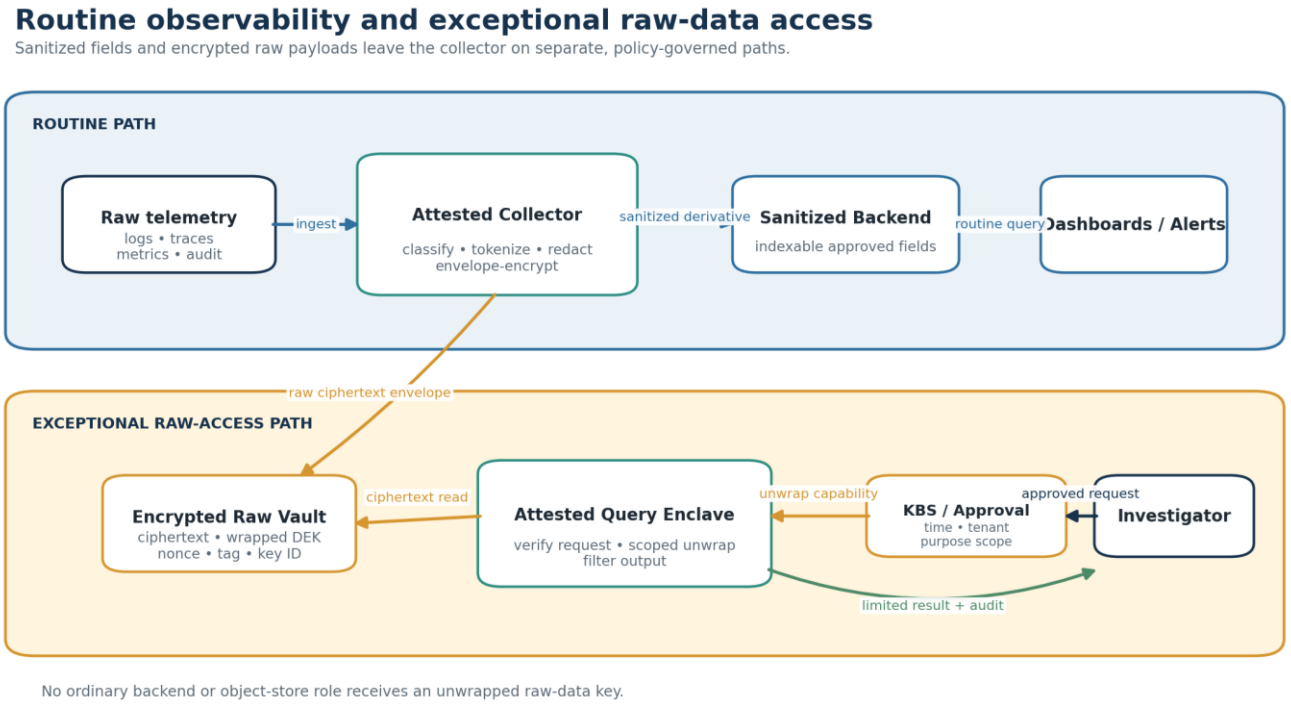


Figure 5. Separation of routine sanitized observability from exceptional raw-data access.

Figure 5 shows the separation between routine observability and exceptional raw access. The conventional backend receives only sanitized indexable fields, while raw content remains ciphertext in the vault. An approved investigator obtains a limited result through an attested query enclave; backend administration and object-store access alone do not constitute decryption authorization.

6.7 Lineage and integrity metadata

Each processed record receives lineage metadata:

```
{
  "confobs.collector.image_digest":                    "sha256:...",
  "confobs.collector.config_digest":                    "sha256:...",
  "confobs.pipeline.manifest_digest":                   "sha256:...",
  "confobs.policy.digest":                              "sha256:...",
  "confobs.attestation.result_id":                      "...",
  "confobs.processing.epoch":                           "2026-05-25T00",
  "confobs.transform.profile":                          "restricted-log-v3",
  "confobs.raw_payload.encrypted":                      true
}
```

The confidential collector signs or MACs batches before export. This helps downstream systems verify that telemetry was processed by an approved collector under an approved pipeline manifest and policy. A prototype must define the batch canonicalization, signature scope, key rotation, verification API, and treatment of partial batches or retries; the design does not claim those mechanisms have already been implemented.

7 Security Analysis

7.1 Confidentiality of raw telemetry

Raw telemetry is protected after it enters a TEE-backed collector, subject to the selected platform's threat model and correct operation. Confidential Containers' trust model, for example, assumes a guest that does not trust the host and maps pods to confidential VMs [12]; a process-enclave deployment would have a different boundary and I/O path.

The main limitation is the ingestion boundary. Direct OTLP and sidecar modes provide stronger protection than node-file collection because host-visible log files may expose data before collector ingestion. Therefore, strict deployments should treat file-based collection as transitional.

7.2 Integrity of pipeline configuration

A malicious administrator might deploy a collector image that disables redaction or exports raw telemetry. ConfObs-TEE mitigates this by tying key release to the collector image, configuration, policy, signed-manifest identity, and platform appraisal. If any approved input changes, the appraisal should fail and the KBS should refuse to release export credentials or decryption capabilities. The implementation must demonstrate that each digest is cryptographically bound into the evidence or attested channel; configuration in an unattested Kubernetes object is insufficient.

7.3 Secret minimization

Secrets are not injected as ordinary Kubernetes Secrets into collector pods. Instead, they are released after attestation and are short-lived. This reduces exposure to etcd compromise, pod-spec inspection, misconfigured RBAC, or host-filesystem access, subject to the selected TEE boundary and the KBS client's handling of plaintext secrets.

7.4 Backend exposure reduction

Conventional observability backends receive sanitized records, tokens, or encrypted payloads. Backend administrators can still perform dashboards, alerts, SLO calculations, and incident triage on preserved operational fields, but they do not automatically receive raw payloads or stable identifiers.

7.5 Multi-tenant isolation

Tenant separation is enforced through namespace-scoped collectors, per-tenant keys, policy-scoped KBS release, and restricted metadata enrichment. A collector for one tenant cannot request another tenant's tokenization or raw-vault capability unless its attestation identity and policy scope permit it.

7.6 Resistance to telemetry poisoning

Confidential computing does not make telemetry truthful. A malicious workload can emit misleading logs, forged attributes, or excessive high-cardinality labels. ConfObs-TEE mitigates this with schema validation, source identity, rate limits, namespace quotas, and signed lineage, but application-level truthfulness remains out of scope.

7.7 Side-channel and platform risks

TEEs reduce reliance on privileged software but do not eliminate all hardware and side-channel risk. Prior research has documented attacks and limitations in TEE systems, including side-channel and fault-injection attacks [15], [16]. FABRICKED demonstrates that platform initialization and firmware correctness can be security-critical for confidential virtual machines [17], [18].

Operational mitigations include:

- Requiring current firmware and microcode;
- Checking TCB status during attestation;
- Revoking keys for vulnerable platform states;
- Minimizing long-lived plaintext inside TEEs;
- Padding or batching sensitive flows where traffic analysis matters;
- Using constant-time cryptographic routines;
- Rotating keys after platform advisories; and
- Treating TEE compromise as a key-revocation event.

These controls reduce risk but do not prove immunity from unpublished platform vulnerabilities.

7.8 Availability risks and fail-closed behavior

Fail-closed attestation can interrupt telemetry. A KBS outage, verifier failure, or platform-attestation issue may prevent collectors from exporting data. The architecture should support:

- Local encrypted buffering without export credentials;
- Redundant KBS and verifier replicas;
- Pre-authorized short-lived credential overlap;
- Explicit alerting when confidential guarantees are unavailable; and
- An optional, separately authorized sanitized emergency path that cannot receive restricted fields or raw-vault capabilities.

The optional emergency path is not an automatic downgrade. It requires a distinct policy, destination, credential class, and data-class allowlist established before the incident. A collector that cannot attest receives no confidential keys or normal exporter credentials; it either remains in quarantine, buffers ciphertext using still-valid scoped material, or sends only data already classified as non-sensitive through the separately authorized path. Silent fallback to plaintext export would violate R8 and the core security model.

8 Implementation Blueprint

8.1 Kubernetes deployment

A practical deployment contains:

- A RuntimeClass for confidential pods;
- An Attested Collector Agent DaemonSet or sidecar;
- A Confidential Gateway Deployment;
- A KBS and verifier service;
- A policy compiler and signed policy bundle;
- A signed pipeline manifest and reference-value registry;
- Namespace-scoped RBAC for Kubernetes metadata enrichment;
- Network policies that allow telemetry only through approved collectors;
- A raw-telemetry vault with encrypted object storage; and
- A sanitized observability backend.

Confidential Containers can provide a cloud-native confidential-runtime foundation. It is designed to enable confidential computing for containers and integrates with Kubernetes [11]–[13]. A prototype must select and document the concrete TEE, runtime, Kubernetes distribution, device and file-ingress path, and supported attestation claims.

8.2 Collector configuration profile

A representative confidential-collector profile includes:

```
receivers:
 otlp:
  protocols:
   grpc:
   http:
 filelog:
  include:
```

```
    - /var/log/pods/*/*/*.log
  kubeletstats:
   collection_interval: 30s

processors:
  memory_limiter:
  k8sattributes:
  transform/confobs_classify:
   # OTTL statements generated from policy bundle
  redaction/confobs:
   # Allow-list and blocked-value patterns generated from policy bundle
  batch:

exporters:
  otlp/sanitized_gateway:
   endpoint: confobs-gateway.observability.svc:4317
   tls:
    cert_file: /confobs/attested/tls.crt
    key_file: /confobs/attested/tls.key
  otlp/raw_vault:
   endpoint: raw-vault-ingest.observability.svc:4317

service:
  pipelines:
   traces:
    receivers: [otlp]
    processors: [memory_limiter, k8sattributes,
                 transform/confobs_classify,
                 redaction/confobs, batch]
    exporters: [otlp/sanitized_gateway]
   logs:
    receivers: [otlp, filelog]
    processors: [memory_limiter, k8sattributes,
                 transform/confobs_classify,
                 redaction/confobs, batch]
    exporters: [otlp/sanitized_gateway, otlp/raw_vault]
   metrics:
    receivers: [otlp, kubeletstats]
    processors: [memory_limiter, k8sattributes,
                 transform/confobs_classify, batch]
    exporters: [otlp/sanitized_gateway]
```

The OpenTelemetry Collector already supports sensitive-data handling through processors [9], but ConfObs-TEE changes the proposed trust boundary: those processors execute after attestation and scoped key release rather than in ordinary host-controlled pods. The configuration above is a profile sketch, not evidence that every named component has been integrated into a confidential runtime.

8.3 Supply-chain binding

The collector image, configuration, policy bundle, and signed pipeline manifest should be signed and reproducibly linked to build provenance. Sigstore Cosign supports signing and attesting container images, and

SLSA provenance describes verifiable information about where, when, and how software artifacts were produced [21], [22].

ConfObs-TEE uses these artifacts as reference inputs for attestation policy. The KBS should not merely verify that a workload is running inside a TEE; it should verify that the workload is the approved collector, running the approved configuration and policy bundle described by an approved manifest. A prototype must specify who signs each artifact, how keys are protected and rotated, how rollbacks are prevented, and how emergency revocation reaches the verifier.

8.4 Workload identity

SPIFFE/SPIRE-style workload identity can complement TEE attestation. SPIRE performs node and workload attestation to issue SVIDs under predefined conditions [20]. In ConfObs-TEE, SPIFFE identity can identify the Kubernetes workload, while TEE attestation supports appraisal of the collector's confidential-execution state and pipeline identity. The two should be combined rather than substituted: an SVID alone does not prove the required TEE state, and TEE evidence alone does not necessarily encode the intended service identity or organizational authorization.

9 Prospective Evaluation and Evidence Plan

This section is a prospective validation plan for a future prototype. It reports no measured results, completed baseline comparisons, acceptance thresholds, or statistical conclusions. The configurations, workloads, metrics, and failure injections below define evidence to collect after a concrete TEE platform, Kubernetes distribution, collector build, policy bundle, and hardware environment have been selected and documented.

Table 4. Prospective research questions and evidence artifacts.

RQ	Focus	Planned variables or mutations	Evidence artifact
RQ1	Confidentiality coverage	Ingestion mode and attempted observation point	Canary-secret exposure matrix with capture method and platform boundary
RQ2	Attestation correctness	Image, configuration, policy, manifest, TCB, freshness, and namespace mutations	Verifier/KBS decisions, credential observations, and quarantine outcomes
RQ3	Performance overhead	Four collector configurations; synthetic and microservice workloads	Throughput, latency, loss, CPU/memory, startup, attestation, and retry measurements with uncertainty
RQ4	Operational utility	Fault scenarios and baseline/sanitized/break-glass views	Diagnostic accuracy, task timing, fields consulted, and approved raw-access use
RQ5	Cost and scalability	Nodes, tenants, pod churn, signal volume, and sampling rate	Resource and latency curves, queue behavior, scaling events, and dated cost basis
RQ6	Failure behavior	KBS, verifier, backend, gateway, node, policy, and network failures	Loss, buffering, recovery, alert visibility, credential expiry, and downgrade-prevention evidence

Table 4 links each research question to an observable artifact rather than a predicted outcome. The four configurations under RQ3 are experimental conditions, not systems that have already been benchmarked. Any eventual result should report hardware and software versions, run counts, warm-up and steady-state intervals, workload distributions, confidence intervals or other uncertainty measures, and all deviations from this plan.

RQ1: Confidentiality coverage

Question. At which points can raw telemetry be observed by host, cluster, backend, and network adversaries?

Method. Instrument a test cluster with controlled canary secrets in logs, spans, metric labels, and audit events. Attempt extraction from:

- Host log files;
- Collector pod logs;
- Collector memory dumps attempted from the host;
- Backend indexes;
- Network captures;
- Retry queues;
- Crash dumps; and
- Raw-vault storage.

Evidence to collect. A matrix showing where canary secrets appear under direct OTLP, sidecar, and node-file collection modes, together with the exact platform threat model and capture method. Absence from one observation point is not proof of absence from all side channels.

RQ2: Attestation correctness

Question. Does key release fail when collector code, configuration, policy, manifest, platform state, or namespace scope changes?

Method. Run mutation tests that:

- Change the collector image digest;
- Modify the redaction policy;
- Modify the signed pipeline manifest;
- Alter an exporter endpoint;
- Remove the confidential runtime;
- Replay old evidence;
- Request a key outside namespace scope; and
- Downgrade platform TCB status if the test infrastructure permits.

Evidence to collect. Verifier and KBS decision logs, evidence identifiers, mutation details, credential observations, quarantine behavior, and pass/fail outcomes for every mutation. This tests the proposed fail-closed property rather than assuming that each digest is bound correctly.

RQ3: Performance overhead

Question. What throughput, latency, CPU, and memory overhead does confidential processing introduce?

Method. Compare four configurations:

- A baseline OpenTelemetry Collector outside a TEE;
- A collector inside the selected TEE without redaction or tokenization;
- A collector inside the TEE with policy processing; and
- The full ConfObs-TEE path with a confidential gateway and raw vault.

Use synthetic OTLP load and a microservices benchmark such as DeathStarBench or Online Boutique. DeathStarBench is an open-source microservices benchmark suite [23], and Online Boutique is a Kubernetes-compatible cloud-first microservices demo [24]. OpenTelemetry publishes Collector benchmark infrastructure that can guide workload generation [25].

Metrics to collect.

- Spans per second, log records per second, and metric datapoints per second;
- P50, p95, and p99 collector latency;
- Dropped records;
- CPU and memory utilization;
- Confidential-workload startup time;
- KBS attestation latency; and
- Export retry and buffering behavior.

No numerical expectation is assigned in advance. The study should report uncertainty and identify whether overhead arises from the TEE runtime, transformations, gateway topology, encryption, or attestation.

RQ4: Operational utility

Question. Does sanitization preserve enough utility for incident response?

Method. Run fault-injection scenarios involving:

- An elevated HTTP 500 rate;
- A slow database dependency;
- A pod crash loop;
- A Kubernetes RBAC misconfiguration;
- Suspicious secret access; and
- A noisy tenant with high-cardinality labels.

Compare mean time to identify root cause using baseline telemetry, sanitized ConfObs-TEE telemetry, and sanitized telemetry plus approved break-glass raw access. The study must predefine tasks, participant expertise, stopping rules, and what constitutes a correct diagnosis before interpreting mean-time differences.

Evidence to collect. Task completion, diagnostic accuracy, fields or views consulted, break-glass requests, investigator comments, and timing with uncertainty. Utility is not established merely because a field remains queryable.

RQ5: Cost and scalability

Question. How does ConfObs-TEE scale with cluster size and telemetry volume?

Method. Vary node count, namespace count, pod churn, log volume, trace-sampling rate, and tenant count. Measure collector autoscaling behavior, KBS and verifier load, key-rotation overhead, and backend index size.

Evidence to collect. Resource and latency curves for each controlled variable, bottleneck observations, scaling events, queue growth, and configuration limits. Monetary cost should be reported only from a documented provider and pricing date.

RQ6: Failure behavior

Question. What happens when attestation, KBS, backend, or gateway components fail?

Method. Inject failures involving:

- KBS unavailability;
- Verifier unavailability;
- Backend unavailability;
- Gateway restart;
- TEE-node drain;
- Policy revocation; and
- Network partition.

Evidence to collect. Data loss, encrypted-buffer growth, alert visibility, recovery time, credential expiry behavior, and evidence that plaintext downgrade did not occur. The separately authorized sanitized emergency path, if implemented, should be tested as its own configuration rather than conflated with fail-closed behavior.

10 Discussion

10.1 Selective confidentiality is preferable to total enclosure

Running an entire observability platform inside confidential-computing environments may be attractive but is rarely practical. Search engines, dashboards, long-term stores, and analytics engines are large, complex, and operationally expensive. ConfObs-TEE instead protects the narrow stages where plaintext and secrets are most exposed, then exports sanitized derivatives. This is an architectural rationale to be tested; the prospective evaluation must determine whether the selected boundary delivers acceptable performance and operational utility.

10.2 The ingestion boundary is the hardest problem

Confidential processing begins only once data enters the TEE. For application logs written to node filesystems, the host may already have seen the data. Therefore, organizations should classify collection modes:

- **Gold:** application emits OTLP directly to an attested collector.
- **Silver:** application and collector run in the same confidential pod.
- **Bronze:** node-local confidential agent reads host-visible files.
- **Legacy:** non-confidential collection with backend-only controls.

This classification, depicted in Figure 4, avoids overstating confidentiality.

10.3 Redaction should be policy-driven, not regex-only

Regex redaction catches common patterns such as tokens or credit-card-like values, but Kubernetes telemetry also contains structured attributes, semantic conventions, labels, and annotations. ConfObs-TEE should combine allowlists, schema-aware rules, regular expressions, endpoint-sensitivity profiles, tenant policies, and application-provided data classifications.

10.4 Metrics are not automatically safe

Metrics are often treated as low risk. This is unsafe in multi-tenant systems. Labels can encode usernames, account IDs, request paths, customer IDs, or internal topology. High-cardinality labels can become privacy leaks and cost hazards. ConfObs-TEE therefore applies classification to all signal types, not only logs.

10.5 Attestation must include configuration

A common mistake is to attest only the runtime environment. For observability pipelines, configuration is security-critical. A single configuration change can disable redaction, add an exporter, expand Kubernetes metadata access, or route restricted logs to a third-party backend. Therefore, collector configuration, policy bundles, and the signed pipeline manifest must be bound into the attested pipeline identity using a mechanism supported by and validated on the selected platform.

10.6 Confidential computing is not a substitute for data minimization

TEEs reduce exposure during processing, but they do not justify collecting excessive telemetry. Data minimization remains the first control. Confidential computing is most valuable when paired with drop-by-default policies, schema controls, short retention, least-privilege export, and audited raw access. Statistical privacy mechanisms such as those used by Prochlo, Prio, or RAPPOR may be more appropriate when aggregate reporting is sufficient [36]–[38].

11 Limitations

This paper has several limitations.

First, the architecture is a design proposal, not a measured prototype result. Future empirical validation should report implementation data from at least one identified TEE platform and one Kubernetes distribution.

Second, the architecture does not eliminate all exposure in file-based log collection. Sensitive workloads should emit directly to attested collectors or use confidential sidecars, and the feasibility of each host-ingress mechanism must be validated for the selected confidential runtime.

Third, TEE guarantees vary by platform, firmware, configuration, vulnerability status, and device path. The KBS must evaluate relevant TCB status and support revocation, while the paper's abstract TEE terminology must not be read as a uniform guarantee across SGX, SEV-SNP, TDX, CCA, and confidential-VM systems.

Fourth, sanitized telemetry may still leak information through metadata, volume, timing, or correlation. Additional controls such as batching, padding, coarse-grained labels, aggregation, or differential privacy may be needed for high-sensitivity tenants.

Fifth, availability is outside the core confidentiality guarantee. Confidential pipelines can fail closed in ways that reduce observability during incidents. This must be managed through redundancy, encrypted buffering, explicit degraded-mode reporting, and carefully separated emergency policy.

Sixth, the signed-manifest, envelope-encryption, lineage, and break-glass mechanisms are specified as design artifacts, not completed implementations. Their schemas, canonicalization rules, cryptographic suites, key custody, approval workflow, and audit semantics require prototype development and security review.

12 Conclusion

Kubernetes observability pipelines process some of the most operationally valuable and security-sensitive data in cloud-native environments. Logs, metrics, traces, and audit events are essential for reliability and security, but they can expose secrets, identities, tenant behavior, internal topology, and regulated information. Existing protections for data in transit and at rest do not protect telemetry while it is being enriched, transformed, sampled, correlated, or exported.

This paper proposes ConfObs-TEE, a confidential-computing architecture for Kubernetes observability data. The design uses attested collector agents,

confidential gateway collectors, policy-bound key release, a defined signed pipeline manifest, tokenization, redaction, envelope encryption, and verifiable lineage to reduce trust in hosts, cluster operators, cloud administrators, and observability backends within a selected platform's documented protection boundary. The key design conclusion is selective protection: raw telemetry and secret-bearing operations should run inside measured TEEs, while sanitized derivatives can continue to use conventional observability platforms.

Future work should implement ConfObs-TEE using OpenTelemetry Collector, Confidential Containers, and an identified production TEE platform; evaluate performance and failure behavior; formalize policy, manifest, envelope, and lineage semantics; and study usability trade-offs for incident-response teams. The broader research direction is that observability pipelines should be evaluated as confidential data-processing systems, not merely as monitoring infrastructure.

References

1. Confidential Computing Consortium, "About Confidential Computing," Linux Foundation, 2026. Available: https://confidentialcomputing.io/about/. Accessed: Aug. 24, 2026.

2. Confidential Computing Consortium, *Confidential Computing: Hardware-Based Trusted Execution for Applications and Data*, Linux Foundation, 2021. Available: https://confidentialcomputing.io/wp-content/uploads/sites/10/2021/03/CCC-A-Technical-Analysis-of-Confidential-Computing-v1.3_unlocked.pdf. Accessed: Aug. 24, 2026.

3. Kubernetes Authors, "Observability," Kubernetes Documentation. Available: https://kubernetes.io/docs/concepts/cluster-administration/observability/. Accessed: Aug. 24, 2026.

4. Kubernetes Authors, "Auditing," Kubernetes Documentation. Available: https://kubernetes.io/docs/tasks/debug/debug-cluster/audit/. Accessed: Aug. 24, 2026.

5. OpenTelemetry Authors, "Collector," OpenTelemetry Documentation. Available: https://opentelemetry.io/docs/collector/. Accessed: Aug. 24, 2026.

6. OpenTelemetry Authors, "Collector Components," OpenTelemetry Documentation. Available: https://opentelemetry.io/docs/collector/components/. Accessed: Aug. 24, 2026.

7. OpenTelemetry Authors, "Install the Collector with Kubernetes," OpenTelemetry Documentation. Available: https://opentelemetry.io/docs/platforms/kubernetes/collector/. Accessed: Aug. 24, 2026.

8. OpenTelemetry Authors, "Important Components for Kubernetes," OpenTelemetry Documentation. Available: https://opentelemetry.io/docs/platforms/kubernetes/collector/components/. Accessed: Aug. 24, 2026.

9. OpenTelemetry Authors, "Handling Sensitive Data," OpenTelemetry Documentation. Available: https://opentelemetry.io/docs/security/handling-sensitive-data/. Accessed: Aug. 24, 2026.

10. H. Birkholz, D. Thaler, M. Richardson, N. Smith, and W. Pan, "Remote ATtestation procedureS (RATS) Architecture," IETF RFC 9334, Jan. 2023. doi: 10.17487/RFC9334.

11. Confidential Containers Authors, "Attestation and Key Broker Service," Confidential Containers Documentation. Available: https://confidentialcontainers.org/docs/architecture/. Accessed: Aug. 24, 2026.

12. Confidential Containers Authors, "Trust Model," Confidential Containers Documentation. Available: https://confidentialcontainers.org/docs/architecture/trust-model/. Accessed: Aug. 24, 2026.

13. Cloud Native Computing Foundation, "Confidential Containers," CNCF Project Page. Available: https://www.cncf.io/projects/confidential-containers/. Accessed: Aug. 24, 2026.

14. OpenTelemetry Collector Contrib Authors, "Kubernetes Attributes Processor," GitHub. Available: https://github.com/open-telemetry/opentelemetry-collector-contrib/tree/main/processor/k8sattributesprocessor. Accessed: Aug. 24, 2026.

15. A. Muñoz, R. Ríos, R. Román, and J. López, "A survey on the (in)security of Trusted Execution Environments," *Computers & Security*, vol. 129, art. 103180, 2023. doi: 10.1016/j.cose.2023.103180.

16. K. Murdock, D. Oswald, F. D. Garcia, J. Van Bulck, D. Gruss, and F. Piessens, "Plundervolt: Software-based fault injection attacks against Intel SGX," in *2020 IEEE Symposium on Security and Privacy*, pp. 1466–1482, 2020. doi: 10.1109/SP40000.2020.00057.

17. B. Schlüter, C. Wech, and S. Shinde, "FABRICKED: Misconfiguring Infinity Fabric to Break AMD SEV-SNP," in *35th USENIX Security Symposium*, 2026. Available: https://www.usenix.org/conference/usenixsecurity26/presentation/schlueter-fabricked.

18. AMD, "SEV-SNP Routing Misconfiguration," AMD Security Bulletin AMD-SB-3034, 2026. Available: https://www.amd.com/en/resources/product-security/bulletin/amd-sb-3034.html. Accessed: Aug. 24, 2026.

19. Open Policy Agent Authors, "Open Policy Agent," Documentation. Available: https://www.openpolicyagent.org/docs/latest/. Accessed: Aug. 24, 2026.

20. SPIFFE Authors, "SPIRE Concepts," SPIFFE Documentation. Available: https://spiffe.io/docs/latest/spire-about/spire-concepts/. Accessed: Aug. 24, 2026.

21. Sigstore Authors, "Signing Containers and Attesting an Image," Sigstore Documentation. Available: https://docs.sigstore.dev/cosign/signing/signing_with_containers/. Accessed: Aug. 24, 2026.

22. SLSA Authors, "Provenance," Supply-chain Levels for Software Artifacts. Available: https://slsa.dev/spec/v1.1/provenance. Accessed: Aug. 24, 2026.

23. Y. Gan et al., "An Open-Source Benchmark Suite for Microservices and Their Hardware-Software Implications for Cloud and Edge Systems," in *Proceedings of ASPLOS*, pp. 3–18, 2019. doi: 10.1145/3293882.3330243. Project: https://github.com/delimitrou/DeathStarBench.

24. Google Cloud Platform, "Online Boutique: Cloud-first microservices demo application," GitHub. Available: https://github.com/GoogleCloudPlatform/microservices-demo. Accessed: Aug. 24, 2026.

25. OpenTelemetry Authors, "Collector Benchmarks," GitHub. Available: https://github.com/open-telemetry/opentelemetry-collector-contrib/tree/main/testbed. Accessed: Aug. 24, 2026.

26. S. Aghili, H. Li, and F. Khomh, "An Empirical Study of Sensitive Information in Logs," arXiv:2409.11313, 2024. Available: https://arxiv.org/abs/2409.11313. Preprint; not peer reviewed as of Aug. 24, 2026.

27. B. Schneier and J. Kelsey, "Secure audit logs to support computer forensics," *ACM Transactions on Information and System Security*, vol. 2, no. 2, pp. 159–176, 1999. doi: 10.1145/317087.317089.

28. D. Ma and G. Tsudik, "A new approach to secure logging," *ACM Transactions on Storage*, vol. 5, no. 1, art. 2, 2009. doi: 10.1145/1502777.1502779.

29. V. Karande, E. Bauman, Z. Lin, and L. Khan, "SGX-Log: Securing system logs with SGX," in *Proceedings of the 2017 ACM Asia Conference on Computer and Communications Security*, pp. 19–30, 2017. doi: 10.1145/3052973.3053034.

30. C. Shepherd, R. N. Akram, and K. Markantonakis, "EmLog: Tamper-resistant system logging for constrained devices with TEEs," in *Information Security Theory and Practice*, LNCS 10844, pp. 75–92, Springer, 2018. doi: 10.1007/978-3-319-93524-9_5.

31. N. Schear et al., "Bootstrapping and maintaining trust in the cloud," in *Proceedings of the 32nd Annual Conference on Computer Security Applications*, pp. 65–77, 2016. doi: 10.1145/2991079.2991104.

32. F. Schuster et al., "VC3: Trustworthy data analytics in the cloud using SGX," in *2015 IEEE Symposium on Security and Privacy*, pp. 38–54, 2015. doi: 10.1109/SP.2015.10.

33. W. Zheng et al., "Opaque: An oblivious and encrypted distributed analytics platform," in *14th USENIX Symposium on Networked Systems Design and Implementation*, pp. 283–298, 2017. Available:

https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/zheng.

34. S. Arnautov et al., "SCONE: Secure Linux containers with Intel SGX," in *12th USENIX Symposium on Operating Systems Design and Implementation*, pp. 689–703, 2016. Available: https://www.usenix.org/conference/osdi16/technical-sessions/presentation/arnautov.

35. D. Vinayagamurthy, A. Gribov, and S. Gorbunov, "StealthDB: A scalable encrypted database with full SQL query support," *Proceedings on Privacy Enhancing Technologies*, vol. 2019, no. 3, pp. 370–388, 2019. doi: 10.2478/popets-2019-0052.

36. A. Bittau et al., "Prochlo: Strong privacy for analytics in the crowd," in *Proceedings of the 26th Symposium on Operating Systems Principles*, pp. 441–459, 2017. doi: 10.1145/3132747.3132769.

37. H. Corrigan-Gibbs and D. Boneh, "Prio: Private, robust, and scalable computation of aggregate statistics," in *14th USENIX Symposium on Networked Systems Design and Implementation*, pp. 259–282, 2017. Available: https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/corrigan-gibbs.

38. Ú. Erlingsson, V. Pihur, and A. Korolova, "RAPPOR: Randomized aggregatable privacy-preserving ordinal response," in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, pp. 1054–1067, 2014. doi: 10.1145/2660267.2660348.

39. M. Mohammady et al., "Preserving both privacy and utility in network trace anonymization," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pp. 459–474, 2018. doi: 10.1145/3243734.3243809.

40. A. Agrawal et al., "DeLog: A high-performance privacy preserving log filtering framework," in *2019 IEEE International Conference on Big Data*, pp. 3785–3794, 2019. Available: https://ieeexplore.ieee.org/document/9006218.