KEYMIRROR

Entrust nShield

Integration Guide

technical@i2km.com

Contact

United States of America
i²km, inc
418 Broadway #4617
Albany
New York
United States of America

Taiwan
i²km company limited
4F-6, Number 61, Yanping South Road
Zhongzheng District
Taipei 100
Taiwan

Contact: support@i2km.com

Changelog

v1.0	18-Sep-2026	Initial

Table of Contents

1 Overview

This guide covers integration of KeyMirror with Entrust general-purpose nShield Hardware Security Modules (HSMs).

Integrating Entrust nShield HSMs with KeyMirror delivers modern observability, tracing and telemetry for HSM client applications, allowing estate-wide observability of cryptographic performance, latencies and algorithms. By seamlessly bridging PKCS#11 into standard enterprise observability stacks, KeyMirror equips organizations to pinpoint issues, prevent outages and rapidly resolve failures. Security and compliance teams gain real-time, audit-ready event streams to satisfy PCI DSS, DORA and SOC 2 mandates without log extraction.

Crucially, KeyMirror allows modern, cloud-native telemetry and tracing to be generated from legacy PKCS#11 applications without code changes, covering both on-premises and cloud HSM use cases. Typical times from installation to observing live telemetry are less than 10 minutes.

KeyMirror fills the client-side observability blind spot for HSM applications, allowing end-to-end monitoring, tracing and alerting. By providing real-time metrics and tracing, operations and security teams can detect bottlenecks, degradation and security anomalies, resolving and mitigating them before they develop into outages and downtime.

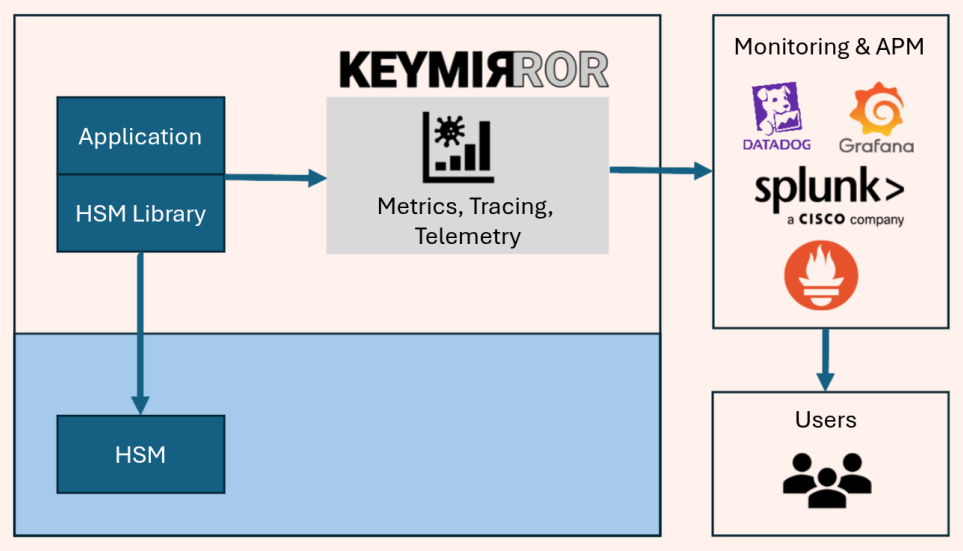


Figure 1: Overview

1.1 Versioning

This guide was created using KeyMirror v1.1 and tested with the following Entrust nShield models and Security World Client images:

Testing Matrix				
Entrust nShield Model	Security World Client Connect Image	FIPS Certified	NIST CMVP	Security World Version
nShield 5C 13.6.14	13.6.12-274-a52430a8	Yes	#5329	v13.6.12 LTS4

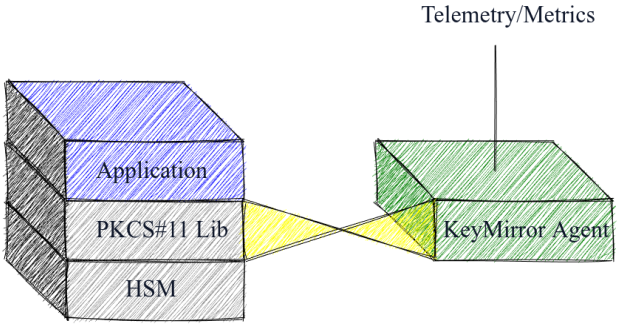
2 KeyMirror Introduction

KeyMirror integrates seamlessly between nShield HSM client applications and libcknfast HSM libraries. KeyMirror captures operational function calls, latency metrics and outcomes without impacting execution or introducing overhead. Extracted events and metrics are normalized into industry-standard formats and delivered to enterprise collectors such as Prometheus, Datadog or Splunk, eliminating complex custom pipelines and enabling instant correlation across existing APM dashboards.

2.1 Deployment Model

KeyMirror instruments client-side HSM applications, hence is deployed in each client environment. A KeyMirror client agent is installed on each client machine from which telemetry is to be collected: these agents do two things:

- They dynamically attach instrumentation to HSM client applications at runtime, without requiring any code changes
- They process the raw observations into OpenTelemetry, then forward to an OpenTelemetry collector, allowing processing, viewing and alerting in standard observability stacks.



2.2 Collection Options

The KeyMirror client agent operates in one of two modes: **Shim Mode** or **BPF Mode**. Determining which mode to use for metadata collection involves a number of considerations.

2.2.1 BPF Mode

In BPF mode, eBPF (Extended Berkeley Packet Filter) probes are used to instrument client applications without modifying process spaces as in Shim Mode. BPF mode involves the KeyMirror client agent attaching kernel probes to PKCS#11 functions in specified libraries or binaries, allowing observation of function data.

In practice, using KeyMirror in BPF mode entails the use of a local KeyMirror configuration file containing the paths of any libraries and/or applications to be observed.

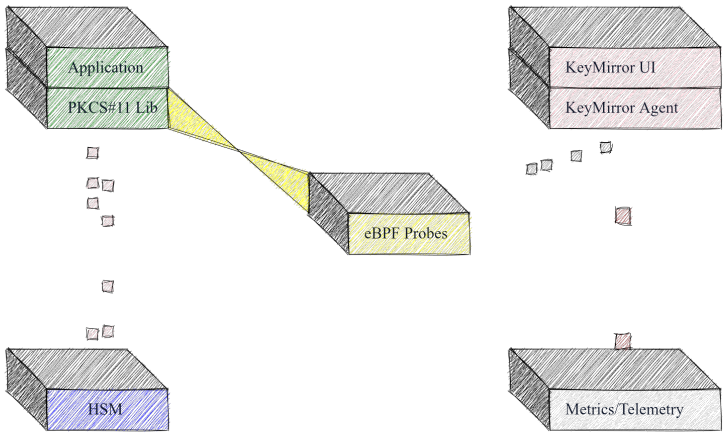


Figure 2: BPF Mode

2.2.2 Shim Mode

In Shim Mode, a shared library (libshim) is interposed between the client application and the libcknfast library. This libshim library presents a standard PKCS#11 interface and observes PKCS#11 function calls, while transparently relaying them to libcknfast. The resulting metadata is then made available to the local KeyMirror client agent which performs processing, converting this into OpenTelemetry metrics and traces, before transmitting this to the configured OpenTelemetry collector.

In practice, using KeyMirror in shim mode entails setting a single environment variable (LD_PRELOAD) or adding an entry to */etc/ld.so.preload* for system-wide usage.

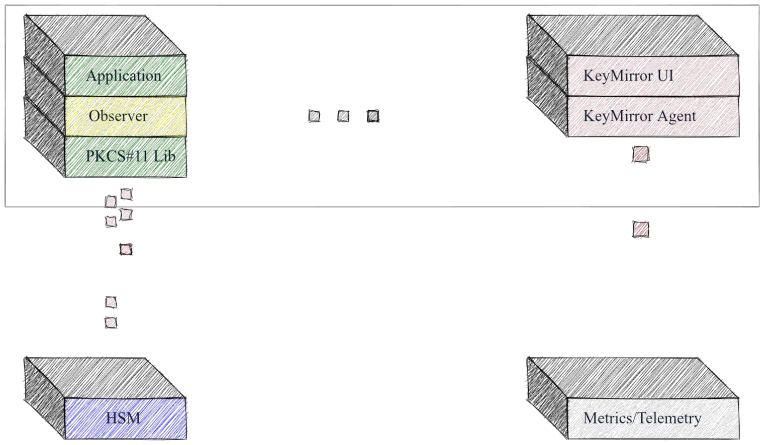


Figure 3: Shim Mode

2.2.3 Choosing and Applying a Mode

While both operating modes deliver the same telemetry, metrics and tracing, choosing between the two modes can be influenced by performance, configuration, operating system and security considerations.

In brief:

- If deployed on Linux and a relatively recent kernel is in use (>= 5.8, late 2020), BPF mode offers negligible runtime overhead, system wide configuration and ability to install the KeyMirror client agent with non-root permissions.
- Similarly, if it is undesirable to restart client processes, e.g. client processes which may have been running for months/years, BPF mode should be used since it requires no application restart
- If KeyMirror is deployed on older Linux versions, BPF mode can have a non-negligible performance impact and/or require root permissions, leading to shim mode being preferable

In general, BPF mode is recommended.

To switch between modes, edit the client agent configuration file (/etc/keymirror/config.json on Linux) and change the “operating_mode” value between “shim” and “ebpf”.[1]

[1] A restart of the KeyMirror client agent is required after an operating mode change

2.3 Headless vs UI Modes

By default, each KeyMirror client agent runs in headless mode, i.e. no UI is available via the client machines. However, for local diagnostic purposes, a local UI can be enabled, allowing a local user to validate the status of the KeyMirror agent, view uptime, number of operations observed and other basic statistics. The local UI is intentionally minimal and text-based, allowing operation in air-gapped environments.

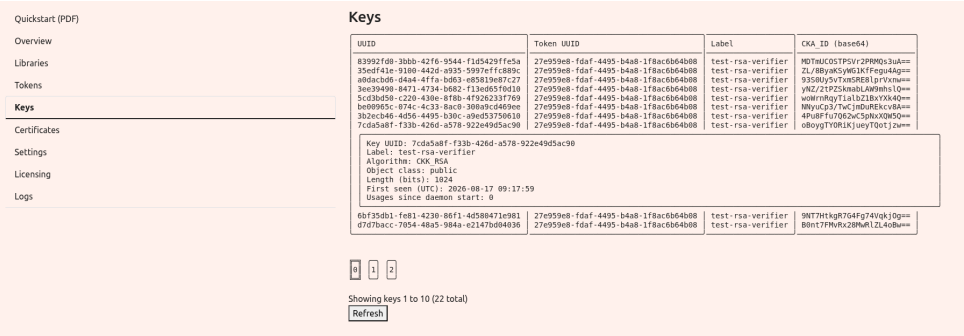


Figure 4: Example Client Agent Diagnostic UI (disabled by default)

2.4 Supported Operating Systems

Version 1.1 of KeyMirror supports Linux environments with BPF mode supporting kernel versions >= 5.8 (2020 era) and shim mode supporting kernel versions >= 4.0 (2015 era).

2.5 Supported nShield Models and Versions

Version 1.1 of KeyMirror has been tested with Security World v13.6.14 and nShield 5c. However, since KeyMirror does not rely on any Security World features or any specific nShield integrations it is expected to be compatible with older versions.[2].

2.6 Supported OpenTelemetry Versions

KeyMirror's use of OpenTelemetry generally obviates versioning considerations. Export to Tempo, Prometheus, Datadog and/or Splunk is handled by configuring the relevant exporters on the OpenTelemetry collector.

[2] In the case that client applications use non-standard PKCS#11 functions or proprietary interfaces such as nCore, KeyMirror does not impede these operations, but capture and representation of these non-standard functions in the collected metrics and tracing is not uniformly available. Contact your representative for more details on supported vendor-specific or non-standard functions.

3 Setting up KeyMirror Client Agents and Telemetry Receiver(s)

Installing KeyMirror agents and monitoring in an estates comprises two essential stages and an optional stage:

1. **Agent install**: On each client machine, install a KeyMirror client agent (e.g. installation of keymirror.deb in Debian-based distributions)
2. **Configure and License KeyMirror's OpenTelemetry Collector Plugin**: OpenTelemetry traffic from KeyMirror client agents is encrypted and OpenTelemetry collectors require a custom receiver to decrypt the traffic from KeyMirror client agents. Installing KeyMirror's OTel receiver plugin allows this traffic to be decrypted when a valid license is applied.
3. **Optional: install the KeyMirror Grafana App**: For ease of use, we publish a Grafana plugin app which allows enhanced viewing, alerting etc of KeyMirror telemetry via Grafana

3.1 Installing the KeyMirror Agent

KeyMirror client agent packages can be downloaded from the KeyMirror client portal (https://portal.i2km.com) and downloading the relevant client package for your architecture[3]. Transfer the package (e.g. keymirror.deb) to the target machine and install (e.g. with "sudo dpkg -i keymirror.deb).

3.1.1 Configuration Options

If further configuration changes are required (beyond the defaults configured in the client portal), edit the agent configuration file at /etc/keymirror/config.json and restart the agent.

3.2 Installing the OpenTelemetry collector plugin

Download the OpenTelemetry receiver plugin from the KeyMirror client portal and follow the instructions on the portal to apply your license and then add the licensed receiver to your OpenTelemetry collector.

[3] Note: the client portal allows configuration options to be specified as defaults, e.g. your OpenTelemetry collector URL, libraries to monitor etc, removing the need to manually configure these on each agent install

4 Integrating nShield HSMs with KeyMirror

Once a KeyMirror client agent has been installed and configured, integrating it with your HSM application is a single-step operation. Following configuration as below, the client agent will gather metadata from the application at runtime, process it and transfer to your OpenTelemetry collector.

4.1 BPF mode configuration

In BPF mode, edit /etc/keymirror/config.json to reflect the application/library path(s) you wish to monitor, followed by restarting the KeyMirror client agent.

For example, to instrument libcknfast, the configuration file would be edited to contain:

```
“files_to_hook“: [“/opt/nfast/toolkits/pkcs11/libcknfast.so“]
```

4.2 Shim mode configuration

In shim mode, launch your application with the environment variable:

```
LD_PRELOAD=/usr/local/lib/liblshim.so
```

4.3 Viewing the Metrics and Tracing

After this has been carried out, the KeyMirror client agent will collect metadata from your application/library at run time, process it, then transfer it to the configured OpenTelemetry receiver. The resulting traces and metrics can then be viewed in Grafana, Datadog etc. For ease of use, we provide a Grafana plugin application, along with example Grafana dashboard files, highlighting the main metrics and tracing behavior. For more details, refer to the screenshots in the appendices.

Appendix A - FAQs

If we deploy KeyMirror, is there any risk of it crashing, causing memory leaks, deadlocks etc?

1. In BPF mode KeyMirror operates via eBPF kernel probes which operate outside of the client/library process and hence are **unable to negatively affect the client application and/or library**. The easiest way to visualize this is that KeyMirror views snapshots of function calls, but without having the ability to impact the application or library. Similarly since KeyMirror operates outside of the existing processes, it's unable to cause locking issues etc.
2. If operating in shim mode, the KeyMirror shim library does operate in the same process space as the client application, leading to a theoretical risk that errors in the shim library could adversely impact the client application. To remove this risk, the KeyMirror shim library makes zero dynamic memory allocations and contains the minimum functionality required to collect and transfer metadata.

What metadata does KeyMirror collect? KeyMirror only collects and reports key/cert labels/IDs, object handles, function call durations and client metadata (e.g. IP address, binary name), but never token passwords, plaintexts, ciphertexts or key values. Object labels and/or IDs can be tokenized if needed.

Does KeyMirror metadata collection affect FIPS boundaries? No. KeyMirror collects metadata outside of FIPS boudaries hence does not affect FIPS certification

Does KeyMirror work with older, air-gapped applications? Yes. KeyMirror is designed to work in air-gapped environments and supports 2014-era onwards environments.

Can we audit the KeyMirror metadata collector code? Yes - we can make our shim and BPF code available for audit with signed, reproducible builds

Has KeyMirror been built with AI? No - KeyMirror is an AI-free product, with every line of code being human-written, ensuring we have full understanding and confidence in our products, minimal technical debt and codebase stability. We only employ AI in a limited manner for security testing, benefiting from AI's security and testing capabilities.

Does KeyMirror metadata collection introduce latency? KeyMirror metadata collection does not add any detectable latency (on the order of nanoseconds)

Do KeyMirror client agents consume significant resources on client machines? No. KeyMirror client agents are highly optimized, typically consuming <50MB RAM, and often little enough that the entire agent can reside in CPU cache. In CPU terms, the KeyMirror client agents are capable of handling >10,000 events per second without significant CPU usage. KeyMirror client agent packages are between 10-15MB in size.

Do KeyMirror client agents contain or rely on third-party packages? KeyMirror client agents have minimal dependencies. Notably, nanopb[4] is employed, along with around 20 golang dependencies, most being OpenTelemetry, Protobuf and logging dependencies.

Appendix B - Viewing Metrics and Telemetry in Grafana

This appendix section covers example screenshots from the KeyMirror Grafana plugin app, along with an example Grafana dashboard.

4.4 Overview of Metrics

The Grafana plugin and example dashboard contain the following sections:

- Sessions: covers open sessions, session activity, traces, hung sessions etc
- Application Errors: presents details of any application errors estate-wide
- Key Usages: presents details of all keys used estate-wide
- Operations: covers operation rates (e.g. how many decrypts/s), latency of operations etc
- Quantum Readiness: covers quantum vulnerable key usages and key creations
- Client Applications: covers application monitoring including memory leak detection

[4] https://jpa.kapsi.fi/nanopb/. NanoPB is also employed in iOS and Android

4.4.1 Sessions

The sessions section displays information regarding all PKCS#11 sessions estate-wide including their number and activity related to each session. Various indicators such as a heatmap of session activity and display of sessions by activity are displayed.

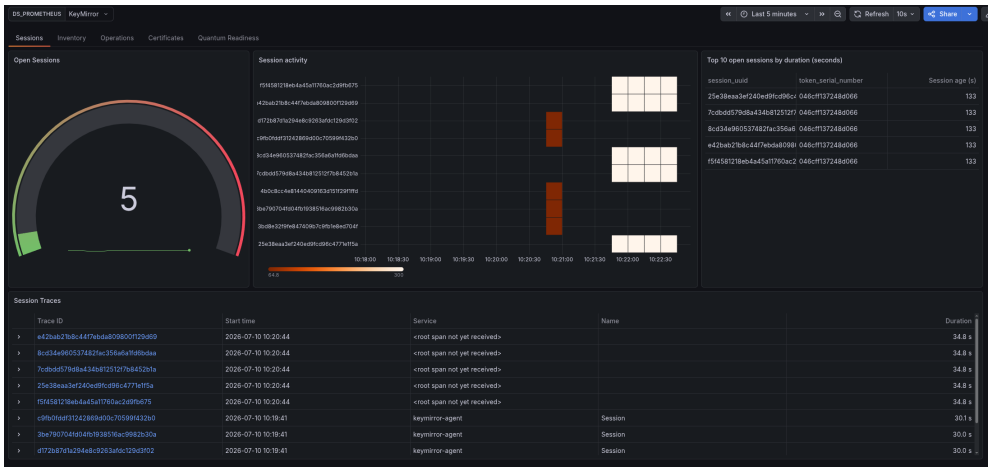


Figure 5: Dashboard - Sessions

4.4.2 Application Errors

The application errors section of the Grafana app presents listing of all application errors which have occurred estate-wide. It categorizes these by session and application, displaying the IP address and binary name, along with the number and type of errors. Clicking the “trace” button for a session will navigate to a trace of the session, allowing the user to view the sequence of events which led to the error(s).

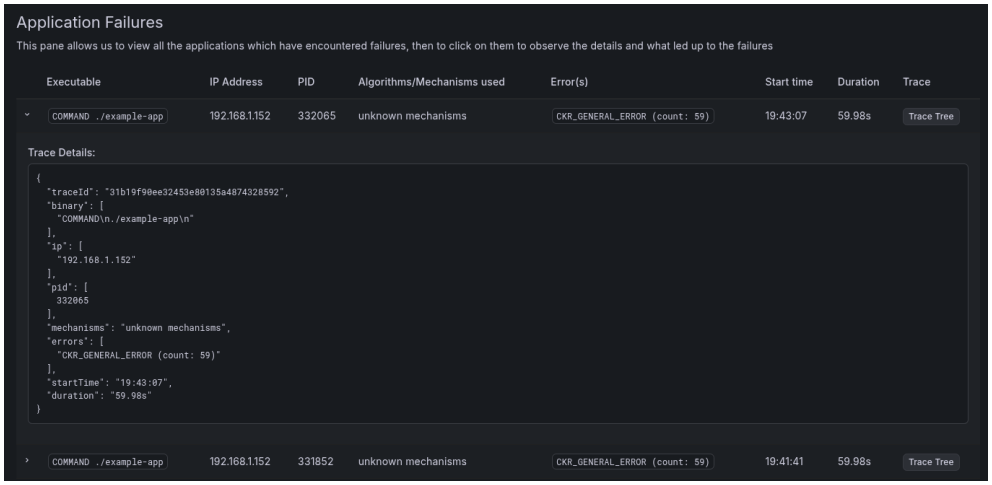


Figure 6: Dashboard - Session Failures

4.4.3 Key Usages

The key usages section of the Grafana app presents listings of all keys used estate-wide in a given timeframe, allowing ranking by usage frequency, latency and algorithm. It allows easy determination of the top-X keys used estate-wide, with the details button providing breakdowns of all applications and client machines accessing a given key, along with providing tracing links to the relevant sessions.

4.4.4 Session Traces

To examine a particular session, click the relevant link in the Session Traces pane. This will display session details and a detailed session trace from the bundled Grafana Tempo service. By expanding the root session trace, information such as the client binary path, client IP address, process ID and operating system can be displayed.

Individual cryptographic operations can also be examined, including their latency and outcome. For example, the following screenshot displays a session trace featuring a number of failed AES-GCM decryption operations.

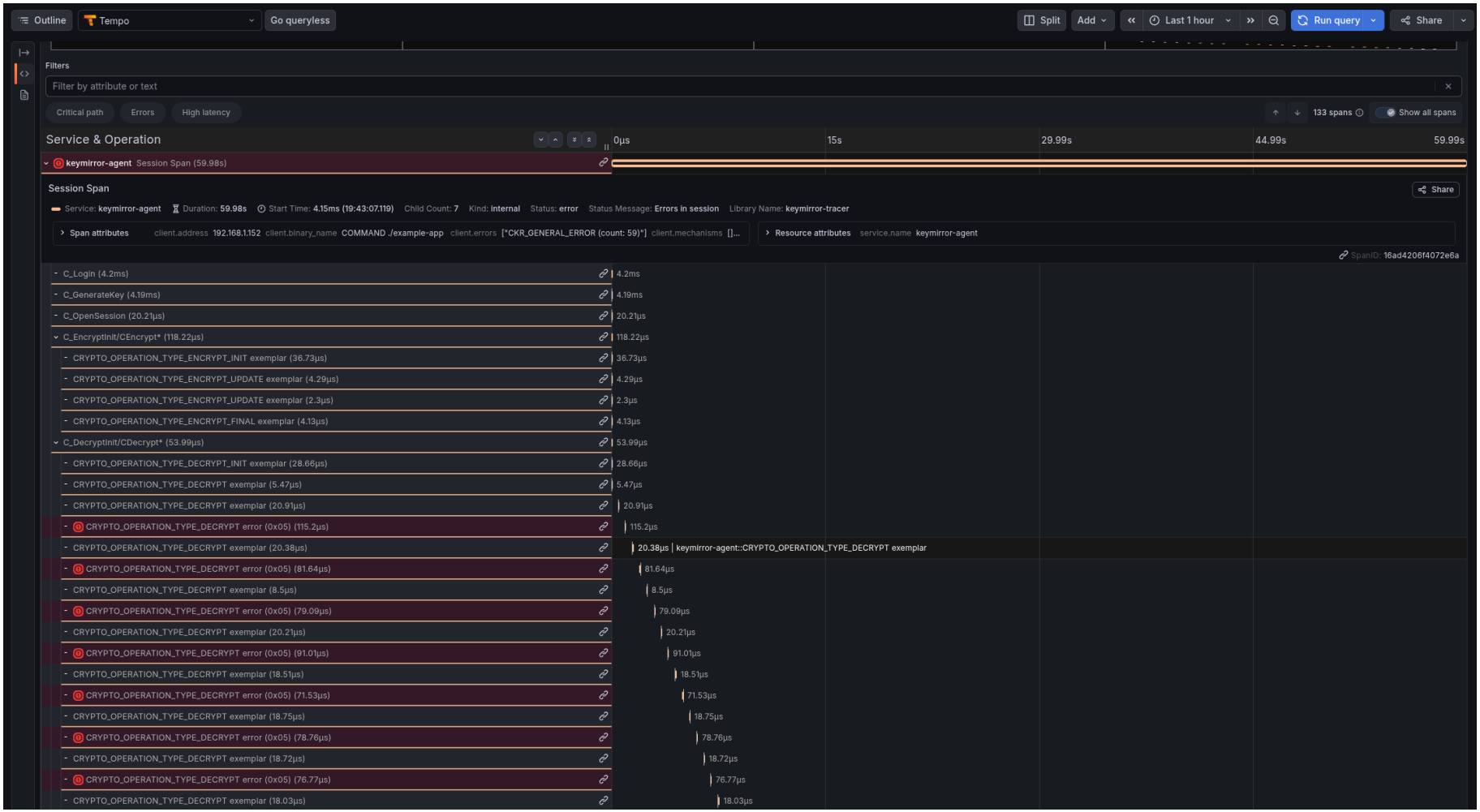


Figure 7: Grafana Dashboard - Example Session Trace

4.4.5 Operations

The Operations tab contains details about the operations, including instantaneous operations rates per second (success and failure), total operations in the time range, a heatmap of key usage by type and a latency histogram.

Figure 8: Dashboard - Operations

4.4.6 Quantum Readiness

The Quantum Readiness tab contains basic details regarding the number of quantum-vulnerable keys generated and used in the time range, allowing basic tracking of quantum posture over time.

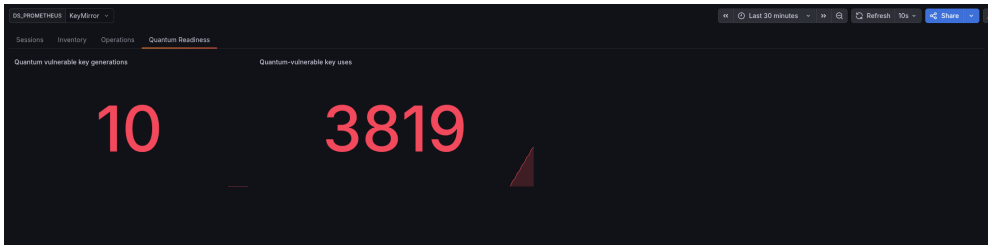


Figure 9: Dashboard - Quantum Readiness

4.4.7 Client Applications

The Client Applications tab contains basic details regarding the applications using the PKCS#11 libraries, including monitoring the memory consumption of each process. The underlying metrics allow detection of memory leaks and alarms to be raised.

Figure 10: Dashboard - Client Applications (memory leak detection)

Appendix C - OpenTelemetry Details

The following is a selection of the OpenTelemetry metrics which KeyMirror produces.[5]

[5] this list is not exhaustive. For a full list, consult the KeyMirror Technical Manual

OpenTelemetry Metric List	
Name	Description
keymirror_open_sessions	Covers the number of open sessions by HSM token and client application IP address
keymirror_abandoned_sessions	Covers the number of abandoned sessions by HSM token and client application IP address
keymirror_idle_sessions	Covers the number of idle sessions by HSM token and client application IP address
keymirror_closed_sessions	Covers the number of closed sessions by HSM token and client application IP address
keymirror_key_generations	Covers the number of key generations by HSM token, client application IP address, session and algorithm
keymirror_key_destructions	Covers the number of key destructions by HSM token, client application IP address, session and algorithm
keymirror_logins	Covers the number of session logins by HSM token, client application IP address and result, allowing failed logins to be monitored and attributed
keymirror_logouts	Covers the number of session logouts by HSM token and client IP address
keymirror_crypto_operations	Covers the number of cryptographic operations by token, session and algorithm
keymirror_latencies	Histograms covering crypto operation latencies by HSM token, session and algorithm
keymirror_session_ages	Covers the age of open sessions by HSM token, session and client application
keymirror_open_find_operations	Covers the number of open find operations by HSM token, session and client application, allowing misconfigured applications to be detected
keymirror_client_mem	Covers the memory consumption of monitored client applications
keymirror_application_errors	Covers the rate of client application errors by client application