# Bundle Protocol Security Library (BSL) Product Guide

---

## DOC-TBA, Prepared by The Johns Hopkins Applied Physics Laboratory

**License**



This work was performed for the Jet Propulsion Laboratory, California Institute of Technology, sponsored by the United States Government under the prime contract 80NM0018D0004 between the Caltech and NASA under subcontract 1700763.

**REVISION HISTORY**

| NUMBER | DATE | DESCRIPTION | NAME |
|---|---|---|---|
| Initial | TBD | Initial issue of document for BSL v1.0.0 | |

# Contents

# List of Figures

# List of Tables

# Introduction

This Product Guide provides architectural and maintenance details about the Bundle Protocol Security Library (BSL), which is part of the NASA Advanced Multi-Mission Operations System (AMMOS) suite of tools.

## Identification

| Property | Value |
|---|---|
| Configuration ID (CI) | 681.4 |
| Element | Mission Control System (MCS) |
| Program Set | Bundle Protocol Security Library (BSL) |
| Version | 1.0 |

## Scope

This document describes technical details about the BSL installation, upgrade, monitoring, and maintenance. For details about the application programming interface (API) and workflows of the BSL see the BSL User Guide.

## Terminology

**Bundle Protocol (BP)**

The overlay network protocol used to transport BPSec blocks and target blocks between nodes.

**Bundle Protocol Security (BPSec)**

The mandatory-to-implement security mechanism to protect blocks of a BP bundle. This is the principal scope of behavior implemented in the BSL.

**BP Agent (BPA)**

The instantiation of a BP node with a unique administrative Endpoint ID.

**BP Endpoint**

The source or destination of a BP bundle, identified by a BP Endpoint ID (EID).

**BP Endpoint ID (EID)**

The identifier of a BP Endpoint; names the source and destination for a BP bundle.

**Host**

A single node on the network and a single instance of an operating system. One host can have many interfaces and many IP addresses, but only one canonical host name.

**Concise Binary Object Representation (CBOR)**

TBD

**JavaScript Object Notation (JSON)**

TBD

## References

| Title | Document Number |
|---|---|
| Software Development | 57653 rev 10 |

Table 1: Applicable JPL Rules Documents

| Title | Document Number |
|---|---|
| MGSS Implementation and Maintenance Task Requirements | DOC-001455 rev G |
| BSL Architecture Description Document | DOC-005089 |
| BSL Software Requirements Document | DOC-005735 |
| BSL Software Interface Specification | DOC-TBD |
| BSL User Guide | DOC-TBD |

Table 2: Applicable MGSS Documents

| Title | Reference |
|---|---|
| BSL Source | GitHub project BSL |
| BSL Documentation Source | GitHub project BSL-docs |
| BSL API Documentation — Main Branch | GitHub Pages for BSL |
| Programming Languages — C | ISO/IEC 9899:1999 |
| IEEE Standard for Information Technology - Portable Operating System Interface (POSIX®) | IEEE Std 1003.1-2008 |
| Security Requirements for Cryptographic Modules | NIST FIPS 140-3 |
| Using SELinux | RHEL9 SELinux Documentation |
| Packaging and distributing software | RHEL9 Packaging Documentation |
| Fedora Packaging Guidelines | Fedora Packaging Documentation |
| OpenSSL Library | https://openssl-library.org/ |
| Unity Test Library | GitHub project Unity |
| NASA Interplanetary Overlay Networking (ION) software | GitHub project for ION-DTN |
| Wireshark Project | https://www.wireshark.org/ |
| Bundle Protocol Version 7 | IETF RFC 9171 |
| Bundle Protocol Security (BPSec) | IETF RFC 9172 |
| Default Security Contexts for Bundle Protocol Security (BPSec) | IETF RFC 9173 |

Table 3: Applicable Other Documents

# Chapter 1

# BSL Architecture

The BSL is purposefully designed to be a software library independent of any specific Bundle Protocol Agent (BPA) implementation and runtime environment. It is intended to be linked to and used by a BPA during runtime to process BPSec security blocks according to local security policy.

The location of the BSL as a subsystem within a BP Node, operated by a BPA is shown in Figure 1.1. The interactions between the BSL and BPA are twofold: calls into the BSL to provide its security services, and calls from BSL into the BPA to provide agent, bundle, and block data and metadata.

Additionally, BSL security services are needed at four distinct points during bundle processing procedures within the BPA. These are depicted in Figure 1.2 and correspond to the following

- After bundle creation from an application source, augmenting the Transmission procedure of [RFC9171].
- Before bundle delivery to an application destination, augmenting the Delivery procedure of [RFC9171].
- After bundle reception via a CLA, augmenting the Reception procedure of [RFC9171].
- Before bundle forwarding via a CLA, augmenting the Forwarding procedure of [RFC9171].

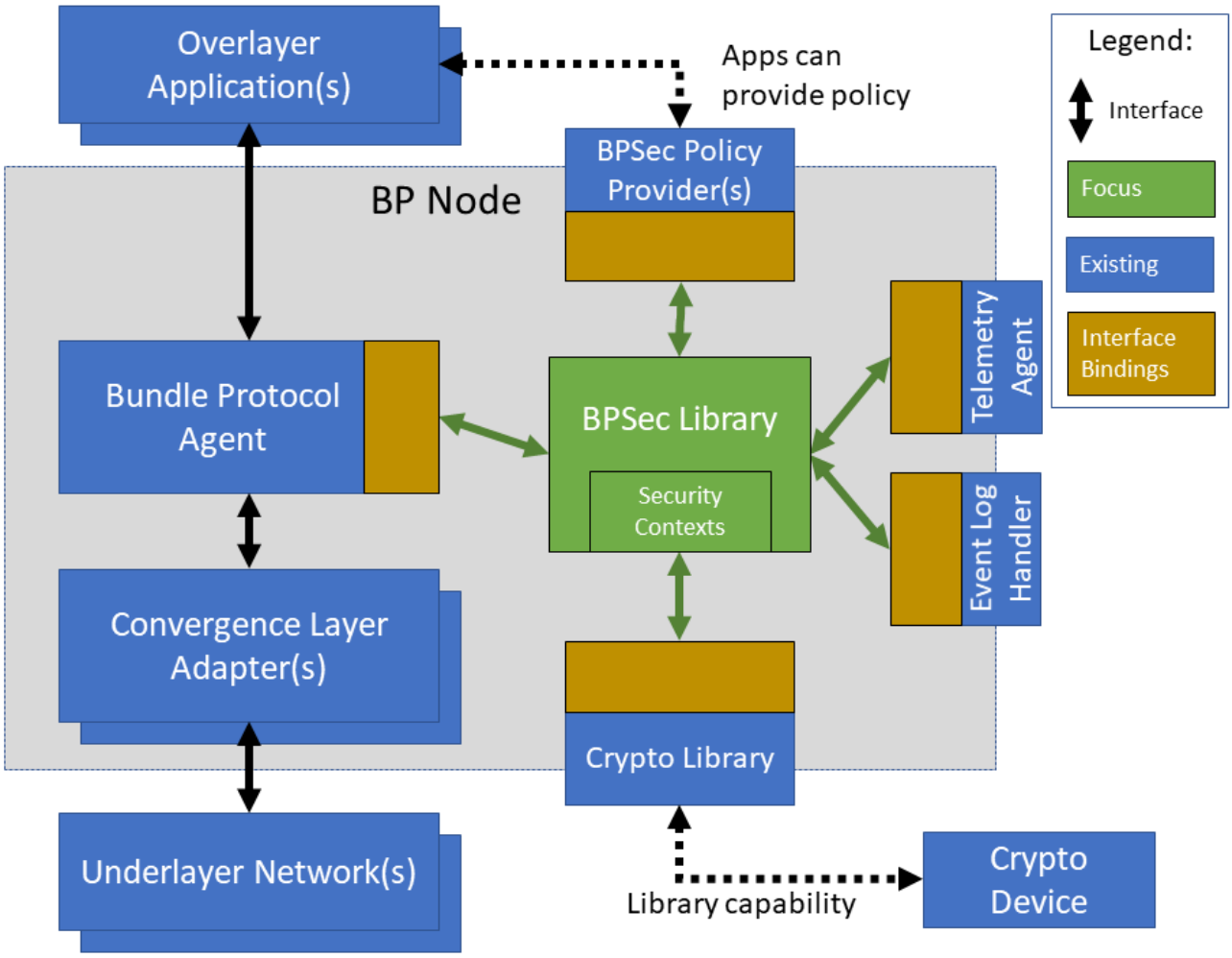


Figure 1.1: BSL System Context

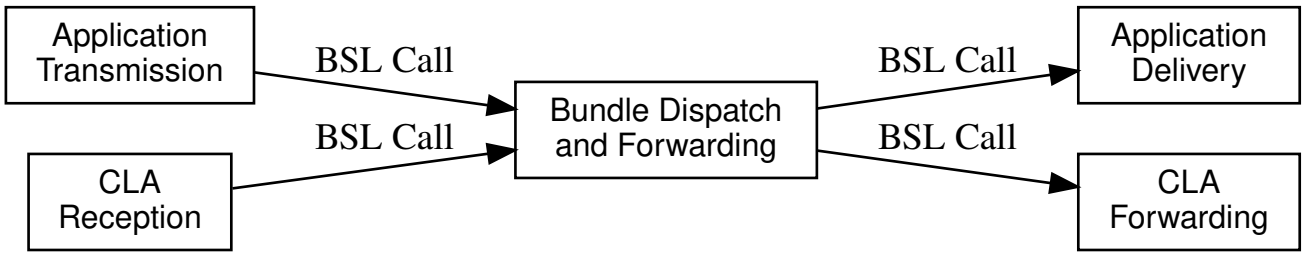


Figure 1.2: Interaction Points from the BPA into BSL

## 1.1 BSL Components

The BSL source is separated into several different components, each of which is explained in detail in the inline API Documentation [bsl-main-api]. A summary of the components is below.

**BSL Frontend** A C99 library used by a BPA integration and used by each Policy Provider and Security Context to access BSL and BPA behavior and data. This is the base of the BSL and is intended to be common for all deployments.

**Dynamic Backend** An implementation of the frontend suitable for general-purpose, non-constrained deployments which uses heap-allocated, dynamically-sized data structures and runtime registration of policy providers and security contexts. This component can be replaced by a deployment-specific alternative if needed.

**Example Policy Provider** An implementation of a configurable policy provider based on the syntax and semantics of the BPSec configuration from the NASA ION software suite [NASA-ION].

**Default Security Contexts** Implementations of the two Default Security Contexts (Context ID 1 and 2) from [RFC9173] using cryptographic functions provided by the OpenSSL library [OpenSSL].

**Mock BPA** An executable used to provide a test fixture and example BPA integration. This Mock BPA does not provide any of the normal processing required of a real BPA by [RFC9171], it is limited to decoding and encoding BPv7 protocol data unit (PDU) byte strings, processing specific BPv7 primary block fields, providing BSL-required integration callbacks, and calling into the BSL for each bundle being processed at each interaction point.

## 1.2 Build and Runtime Environments

The basic requirements in the BSL SRD are that the build environment use a C compiler, with its standard headers and libraries [C99], and include POSIX headers and libraries [POSIX].

The example ION-heritage policy provider distributed with the BSL uses the TBD library for JSON parsing.

The example security contexts distributed with the BSL uses the [OpenSSL] library for all cryptographic functions.

The Mock BPA distributed with the BSL uses POSIX UDP/IP sockets for BPv7 PDU transport, both as a test CLA and a test application interface. This allows traffic into and out of the Mock BPA to be captured by tools such as `pcap` and inspected with tools such as Wireshark and `tshark` [wireshark].

Unit tests for each of the BSL components use the [unity-test] library for defining test fixtures and assertion logic.

## 1.3 Software Packaging

The official releases of the BSL are packaged and distributed as RPM packages intended to be usable within a YUM/DNF repository [rhel9-packaging]. Packages are version marked based on the latest git tag in the working copy's commit history and revision marked based on the specific latest git commit hash of the working copy along with the distribution tag (see the "Versioning" and "Dist Tag" sections of [fedora-packaging]).

For example, a pre-release build of the BSL is marked with RPM version-revision of `0.0.0-0.g71ab437.el9` indicating it does not follow a release version tag (so gets marked with version `0.0.0`), it is zero commits from that (non-)tag, it is from commit hash `71ab437`, and it was built on RHEL-9 (or equivalent).

BSL packages can also built from the source tree, either under RHEL-9 directly or using a (Docker or Podman) container to provide an RHEL-9 environment. Details on these procedures are provided in Section 2.1.

The set of packages for each BSL release (or local package build) contains the following:

**`bsl`** The runtime files needed for the library itself. This contains versioned shared objects. Major files are installed under `/usr/lib64/`.

**`bsl-devel`** Development files needed to build and link against the BSL. This contains C headers and shared object version links. Major files are installed under `/usr/include/` and `/usr/lib64/`.

**`bsl-apidoc`** Doxygen-generated API documentation derived from in-source markup. Major files are installed under `/usr/share/d` which contains an `html` directory.

**`bsl-debuginfo`** Runtime debug information associated with the `bsl` package. This relies on `bsl-debugsource` for tracing to individual source lines for interactive debugging.

**bsl-debugsource** Copies of the original source files used along with the `*-debuginfo` packages to support interactive debugging.

**bsl-test** Major files are installed under `/usr/bin/`, containing the `bsl-mock-bpa` executable, `/usr/lib64/` for its libraries, and `/usr/libexec/bsl/` which contains each unit test executable for the BSL.

**bsl-test-devel** Development files needed to build and link against the Mock BPA of the BSL. This contains C headers and shared object version links, including the Unity test library. Major files are installed under `/usr/include/` and `/usr/lib64/`.

**bsl-test-debuginfo** Runtime debug information associated with the `bsl-test` package. This relies on `bsl-debugsource` for tracing to individual source lines for interactive debugging.

## 1.4 File System

The BSL itself does not require any specific input or configuration files for its normal operation. It relies on the host BPA to perform any configuration file management, loading, parsing, *etc.*.

As a Linux shared library, it does relate to the host file system in the following paths:

**/usr/lib64/** The OS-standard path for all shared library files. The BSL installs its core and example libraries here.

**/usr/include/** The OS-standard path for all library header files. The BSL installs its own headers under the `bsl` sub-directory, and its inbuilt (non-OS) dependencies under `QCBOR` and `m-lib` sub-directories.

**/usr/bin/** The OS-standard path for all non-privileged executable files. The BSL installs its Mock BPA as the executable `bsl-mock-bpa` here.

**/usr/libexec/** The OS-standard path for context-dependent executable files. The BSL installs its unit tests under the `bsl` sub-directory.

## 1.5 Networking

The BSL itself does not require any specific OS networking configuration or API interfaces. It relies on the host BPA to perform any network configuration or runtime use.

The Mock BPA distributed with the BSL uses UDP/IP sockets, configured by command-line options, to communicate bundles into and out of the BPA process.

## 1.6 Cryptographic Functions

The BSL itself does not require any specific OS or middleware cryptographic functions.

The example implementation of the default security contexts distributed with the BSL uses the [OpenSSL] library for performing all cryptographic functions.

# Chapter 2

# Procedures

This chapter includes specific procedures related to managing an BSL deployment from source and for development of BSL changes.

## 2.1 Building Packages

The BSL source is composed of a top-level repository `BSL` [bsl-source] and a number of submodule repositories; all of them are required for building the BSL.

1. The top-level checkout can be done with:

   ```
   git clone --recursive --branch <TAGNAME> https://github.com/NASA-AMMOS/BSL.git
   ```

2. Optional: switching to a different tag or branch can be done with the sequence:

   ```
   git checkout <TAGNAME>
   git submodule update --init --recursive
   ```

3. More TBD

## 2.2 Installation

TBD

## 2.3 Upgrading

Because the BSL is deployed in an RPM package form, the normal operating system tools and procedures for dealing with software library upgrading apply to the BSL. The BSL provides SOVERSION information in its libraries, so RPM management tools such as DNF which are cross-dependence-aware will ensure that the correct needed SOVERSION of the BSL is installed.

Individual BSL releases may identify pre-upgrade or post-upgrade steps in their specific Release Description Document (RDD) which would augment this OS-standard procedure.

## 2.4 Development Building

## 2.5 Monitoring

The BSL itself, as a software library, does not directly make use of any logging mechanism.

TBD

### 2.5.1 SELinux Audit Events

The procedures in this section are a summary of more detail provided in Chapter 5 of the RedHat [rhel9-selinux] document.

By default, the `setroubleshootd` service is running, which intercepts SELinux audit events

To observe the system audit log in a formatted way run:

```
sudo sealert -l '*'
```

Some SELinux denials are marked as "don't audit" which suppresses normal audit logging when they occur. They are often associated with network access requests which would flood an audit log if they happen often and repeatedly. To enable logging of `dontaudit` events run:

```
sudo semanage dontaudit off
```

## 2.6 Checkout Procedures

The BSL packaging procedure includes built unit tests within the `bsl-test` RPM package which allows executing unit tests on the BSL library after build time on any other host.

TBD

# Chapter 3

# Product Support

There are two levels of support for the BSL: troubleshooting by a system administrator, which is detailed in Section 3.1, and upstream support via the BSL public GitHub project, accessible as described in Section 3.2. Attempts to troubleshoot should be made before submitting issue tickets to the upstream project.

## 3.1 Troubleshooting

### 3.1.1 Installation

This section covers issues that can occur during installation (see Section 2.2) of the BSL.

TBD

### 3.1.2 Operations

This section covers issues that can occur after successful installation (see Section 2.2) and checkout (see Section 2.6) of the BSL.

### 3.1.3 SELinux Blocked Behavior

TBD

### 3.1.4 FIPS-140 Blocked Behavior

TBD

#### 3.1.4.1 TBD

## 3.2 Contacting or Contributing

The BSL is hosted on a GitHub repository [bsl-source] with submodule references to several other repositories. There is a `CONTRIBUTING.md` document in the BSL repository which describes detailed procedures for submitting tickets to identify defects and suggest enhancements.

Separate from the source for the BSL proper, the BSL Product Guide and User Guide are hosted on a GitHub repository [bsl-docs], with its own `CONTRIBUTING.md` document for submitting tickets about either the Product Guide or User Guide.

While the GitHub repositories are the primary means by which users should submit detailed tickets, other inquiries can be made directly via email to the the support address dtnma-support@jhuapl.edu.