PNNL-28310

# Keyless Infrastructure Security Solution (KISS): VOLTTRON™ KSI® Blockchain Design and Specification

Cybersecurity for Energy Delivery Systems (CEDS) Research and Development

December 2018

M Mylrea
SN Gourisetti
V Tattireddy, Guardtime
K Kaur, Washington State University
C Allwardt
R Singh
J Plummer, Guardtime
R Bishop, Guardtime
A Hahn, Washington State University

Prepared for the U.S. Department of Energy
under Contract DE-OE0000190

PACIFIC NORTHWEST NATIONAL LABORATORY
*operated by*
BATTELLE
*for the*
UNITED STATES DEPARTMENT OF ENERGY
*under Contract DE-AC05-76RL01830*

This material is based upon work supported by the U.S. Department of Energy Office of Electricity under Award Number M617000254.

PNNL-28310

# Keyless Infrastructure Security Solution (KISS): VOLTTRON$^{TM}$ KSI$^{®}$ Blockchain Design and Specification

Cybersecurity for Energy Delivery Systems (CEDS) Research and Development

December 2018

M Mylrea
SN Gourisetti
V Tattireddy, Guardtime
K Kaur, Washington State University
C Allwardt
R Singh
J Plummer, Guardtime
R Bishop, Guardtime
A Hahn, Washington State University

Prepared for
the U.S. Department of Energy
under Contract DE-AC05-76RL01830

Pacific Northwest National Laboratory
Richland, Washington 99352

## Revision History

| Revision | Date | Deliverable (Reason for Change) | Release # |
|---|---|---|---|
| | 12/15/2018 | Original release | |

## Summary

This document provides the technical design specifications that are required for Keyless Infrastructure Security Solution (KISS) development, including, definitions and roles of the new KISS-related products—GridAware and LinkLite, an overview of the DNP3 communications the VOLTTRON™ DNP3 agent is using for distribution control, data flow, historian processing, setup, and installation. These technical specifications provide step-by-step instructions for configuration, implementation, and deployment of the KISS subsystem, docket creation, and verification using the GridAware dashboard.

Based on this document and the KISS specification requirements and device risk assessment documents, KISS will be developed by integrating keyless signature infrastructure (KSI) blockchain to increase the cybersecurity, integrity, and trustworthiness of critical energy delivery systems (EDSs). The KISS architecture consists of a VOLTTRON™ agent-based distribution control system platform and GridAware (KSI blockchain capability) for secure energy exchanges in a decentralized electrical network.

# Acronyms, Definitions and Abbreviations

| | |
|---|---|
| DMS | distribution management system |
| docket | Guardtime construct that encompasses XDAL and KSI signatures |
| EDS | energy delivery system |
| EMS | energy management system |
| GridAware | Provides security fabric to manage and maintain decentralized networks and devices such as the energy delivery systems |
| ID | identifier |
| JSON | JavaScript Object Notation |
| KISS | Keyless Infrastructure Security Solution |
| KSI | Keyless Signature Infrastructure |
| PNNL | Pacific Northwest National Laboratory |
| RTU | Remote Terminal Unit |
| UUID | universally unique identifier |
| VOLTTRON™ | agent-based distribution control system platform |
| XDAL | eXtensible Data Attribution Language |

## Contents

## Figures

## Tables

# 1.0 Introduction

This document provides technical design specifications for Keyless Infrastructure Security Solution (KISS) software system. KISS will integrate keyless signature infrastructure (KSI) blockchain to increase the cybersecurity, integrity, and trustworthiness of critical energy delivery systems. The KISS consists of VOLTTRON™ agent-based distribution control system platform and GridAware[1] for secure energy exchanges in a decentralized electrical network. This document describes the integration of KSI software system with VOLTTRON™; demonstrates the process of data ingestion, docket creation, and verification; and describes the VOLTTRON™ DNP3 agent communication. The DNP3 agent will be used in the use case to secure the data flow between grid systems such as remote terminal unit (RTU) and Energy Management System/Distribution Management System (EMS/DMS). This document provides detailed user guidelines for deploying and testing the KISS subsystem.

## 1.1 VOLTTRON™ Overview

VOLTTRON™ is an open-source, open-architecture platform that serves as an integration platform for the components of the transactional network. It provides an environment for agent execution and serves as a single-point of-contact for interfacing with distributed devices (e.g., RTUs, building systems, meters, etc.), external resources, and platform services such as data archiving and retrieval. VOLTTRON™ provides a collection of utility and helper classes, which simplifies agent development. VOLTTRON™ connects devices to applications implemented in the platform, a data historian, and signals from the power grid. VOLTTRON™ incorporates a number of open-source products to build a flexible and powerful platform. Referring back to the use cases defined in the deliverables, “D1.2. White Paper Examining Security and Trust Gaps” and “D1.3. Blockchain for Complex Grid Edge Transaction Energy Requirements Document”, the KISS experimentation and testing through the use cases will be performed by using both the simulated model data as well as the data from the test bed grid systems deployed at PNNL and WSU.

## 1.2 GridAware Overview

Guardtime’s GridAware provides a decentralized security fabric for securing, provisioning, monitoring, maintaining, and managing modern decentralized networks and environments such as field area networks, critical infrastructure, SmartGrids, and the Internet of Things. GridAware key components include GridAware data capture agents and services, GridAware support services, GridAware dashboards, and the KSI blockchain. GridAware provides decentralized and distributed security by leveraging Guardtime’s KSI blockchain[2] and provides new

[1] GridAware has previously been referenced as Resonance in KISS documentation. Guardtime changed their product name; therefore, the KISS project and corresponding documentation is proceeding with Guardtime’s revised product-naming convention.

[2] M. Mylrea et al, “Technology Landscape Analysis Report”, PNNL, April 2018. According to the details presented in the landscape report, Keyless Signature Infrastructure (KSI) is Guardtime’s software system that is coupled with their blockchain. The components together are referred to as KSI Blockchain. Extensive details about Guardtime’s KSI technology was discussed in D1.2. White Paper Examining Security and Trust Gaps” and “D1.3. Blockchain for Complex Grid Edge Transaction Energy Requirements Document”.

approaches and mechanisms to ensure granular security, secure device monitoring and management, and data exchange in a “Zero-Knowledge” fabric.

## 1.3 KISS Platform Overview

KISS is based on the Pacific Northwest National Laboratory’s (PNNL) VOLTTRON™ distribution control system integrated with the Guardtime GridAware platform. Together they provide a platform for managing energy exchange that requires security and trustworthiness. Figure 1 provides an overview of the KISS architecture. The main components of KISS architecture are VOLTTRON™ (PNNL developed software platform) and GridAware (Guardtime developed technology). LinkLite is the software component that will be interacting with external services such as VOLTTRON™ message bus. The assets and systems that are participating in the KISS blockchain system will be interacting through the VOLTTRON™ subsystems and respective agents. Appendix A describes the KISS agent set-up.

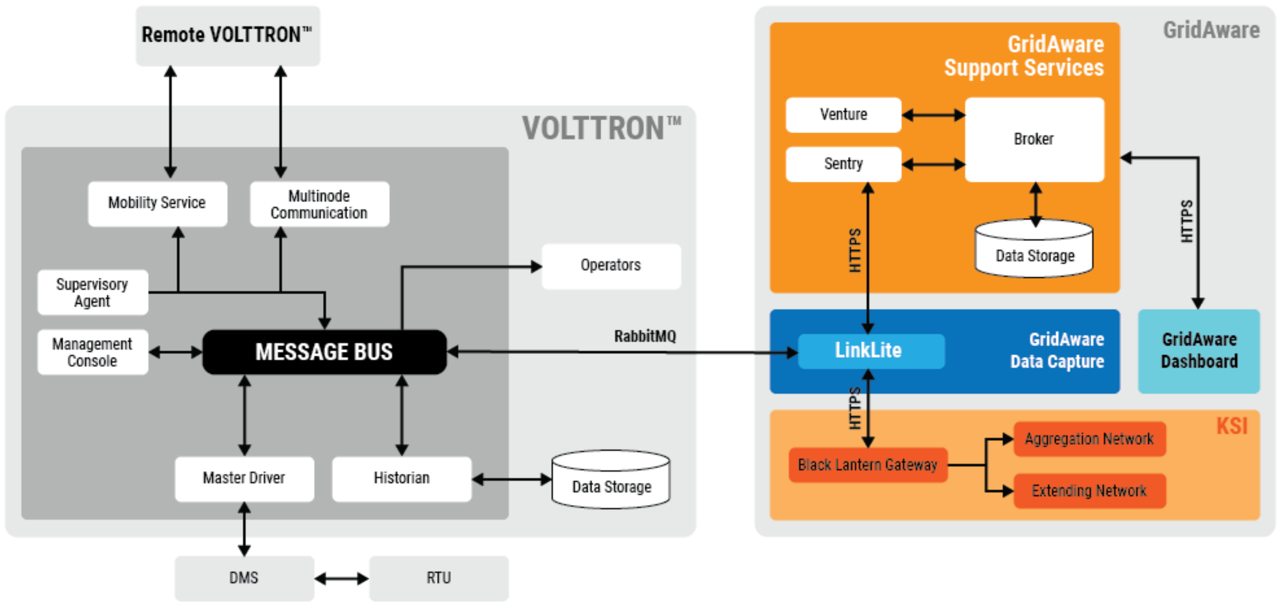


Figure 1. KISS (VOLTTRON™ + GridAware) Architectural Overview

## 2.0 Architectural Design

KISS consists of VOLTTRON™ agent-based distributed control system platform enabled with Guardtime's GridAware for managing the exchange of energy-related data, measurements, and information that requires security and trustworthiness in a decentralized electrical network. Both VOLTTRON™ and GridAware are composed of various functional components. A high-level overview of the KISS components is described in **Error! Reference source not found.**.

Table 1. KISS Components and Their Associations

| Component | Association | Description |
|---|---|---|
| LinkLite | GridAware | GridAware Data Capture Component that interfaces with VOLTTRON™ agents using RabbitMQ |
| Master Driver | VOLTTRON | VOLTTRON™ platform driver that communicates with grid systems such as Distributed Management Systems, Energy Management Systems, etc. and receives data from remote device units |
| Historian | VOLTTRON | VOLTTRON™ platform agent that enables the storage of device data obtained by the drivers and provides data retrieval functions for analysis and support |
| Dashboard | GridAware | GridAware dashboard provides multiple levels of data visualization to the end user about the state of devices and associated agents. |
| Docket | GridAware | A docket is an XML document based on Guardtime's eXtensible Data Attribution Language (XDAL) schema for a data attribution construct, containing data and its KSI signature. |
| Support Services | GridAware | GridAware support services are backend services for policy enforcement, storage, data and event correlation, work flow execution, alerting, and business rule application. |

### 2.1 LinkLite

LinkLite is a GridAware data capture component that provides data capture, docket creation, docket verification, and docket routing to GridAware support services. A docket is an XML construct based on the Guardtime XDAL schema for data attribution. Dockets are the basic building blocks of GridAware components and are used for data normalization, interoperability, policy enforcement, and workflow automation.

LinkLite captures the data from the master driver agent by creating a docket and routing the docket to the GridAware support services or other VOLTTRON™ instances for policy enforcement, storage, workflow execution, and business rule application. LinkLite also provides the VOLTTRON™ instances with data and command verification services for VOLTTRON™ agents. This process facilitates the VOLTTRON™ platform with cryptographically provable and granular local data integrity and detailed policy or rules verification via the docket and KSI blockchain.

LinkLite is a Java-based service that can run on the VOLTTRON™ platform server or a remote server. The LinkLite service provides flexible and configurable interfaces with the VOLTTRON™ platform via RabbitMQ using a Remote-Procedure-Call Application Programming Interface (RPC API)-styled message pattern-Request/Response for publication and consumption of data.

## 2.2 VOLTTRON™ DNP3 Communication

The DNP3 protocol (IEC 60870-5, IEEE 1815-2010) is one of the most widely used protocols in the energy delivery systems (EDSs) environment. It is commonly used in communication between the EMS, DMS, and RTU. Further, the communications between the RTU and protection relays could also be performed using DNP3 protocol. To test the KISS subsystem on a realistic utility-scale use case, the KISS team has defined the use-cases to secure the data exchange and communications between the EMS/DMS and RTU. The KISS team will explore expanding the use cases to other grid systems such as protection relay communication and data exchange with the RTU. Since the VOLTTRON™ message bus is an integral part of the KISS subsystem, the existing VOLTTRON™ DNP3 agent will be used for tests and in system deployment (i.e., both in beta and production software). To have real communication flow within the KISS system, VOLTTRON™'s DNP3 agent is being used in conjunction with the master driver.

For example, the configuration of feeder protection relays has been used. Using an OpenDNP3[3] library, a DNP3 master communicates with the VOLTTRON™ DNP3 agent that acts as an outstation. The DNP3 agent loads empty data points into the database when it starts. The configuration file in the DNP3 driver store has a defined set of data points that get added to the empty points. The agent listens on port 20000 for DNP3 messages and establishes a connection with the DNP3 master. Figure 2 shows the logs when the DNP3 agent is started.

The agent communicates with the VOLTTRON™ master driver, which has a configuration of all the data points from the relays. The data points are stored in a csv file with the format <DNP3 Point Name, VOLTTRON Point Name, Group, Index, Scaling, Units, Writable> shown in Figure 2. The first two parameters determine the data point, and the group and index define the data type and group to which a specific data point belongs. "Writable" is a True/False field that states which values can be written. For instance, Switch Operate Commands for the relays are writable, while the load current or phase voltage are not.

The master driver takes the values of different data points from the DNP3 agent and publishes them onto the message bus with their names and values. Once the data points are published on the bus, they can be sent to LinkLite for signing and verification purposes. Figure 3 shows the data points being published by the VOLTTRON™ master on the message bus.
The values of the data points are being published on the VOLTTRON™ message bus, but it is required for the values to be updated when a write request is sent from the DNP3 master. This capability is currently under development. An overview of DNP3 communication is shown in Figure 4.

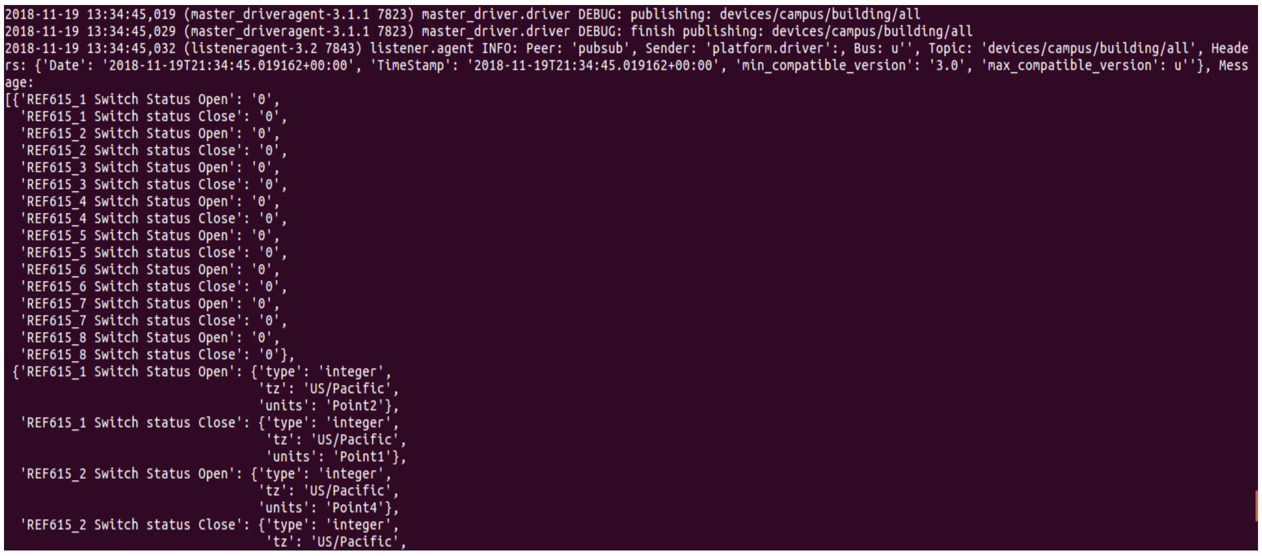
## 2.1 Historian Processing

The VOLTTRON™ historian archives data in persistent storage received from the master driver by verifying policies and data integrity using LinkLite. For KISS use-cases, a small portion of a feeder's historian data is used. These historian data include power system measurements (e.g., voltages at nodes) and logs (e.g., system alerts). The VOLTTRON™ historian will contain the dockets of data from the base historian.

[3] https://github.com/automatak/dnp3

```
2018-11-19 12:51:16,266 (dnp3agent-1.1 3029) dnp3.base_dnp3_agent
INFO: Publishing outstation status: ChannelState.OPENING
2018-11-19 12:51:16,268 (dnp3agent-1.1 3029) dnp3.outstation
DEBUG: DNP3Log TCPServer.cpp(98) (filters=8) Listening on:
0.0.0.0:20000
2018-11-19 12:51:16,268 (dnp3agent-1.1 3029) dnp3.base_dnp3_agent
INFO: Publishing outstation status: running
2018-11-19 12:51:16,282 (dnp3agent-1.1 3029)
volttron.platform.vip.agent.subsystems.configstore DEBUG:
Processing callbacks for affected files: {'mesa_points.config':
'NEW', 'config': 'NEW'}
2018-11-19 12:51:16,282 (dnp3agent-1.1 3029) dnp3.base_dnp3_agent
DEBUG: DNP3Agent configuration parameters:
2018-11-19 12:51:16,282 (dnp3agent-1.1 3029) dnp3.base_dnp3_agent
DEBUG:     points type=<type 'list'>
2018-11-19 12:51:16,282 (dnp3agent-1.1 3029) dnp3.base_dnp3_agent
DEBUG:     point_topic=dnp3/point
2018-11-19 12:51:16,282 (dnp3agent-1.1 3029) dnp3.base_dnp3_agent
DEBUG:     outstation_status_topic=mesa/outstation_status
2018-11-19 12:51:16,283 (dnp3agent-1.1 3029) dnp3.base_dnp3_agent
DEBUG:     local_ip=0.0.0.0
2018-11-19 12:51:16,283 (dnp3agent-1.1 3029) dnp3.base_dnp3_agent
DEBUG:     port=20000
2018-11-19 12:51:16,283 (dnp3agent-1.1 3029) dnp3.base_dnp3_agent
DEBUG:     outstation_config={'link_local_addr': 101,
'database_sizes': 10000, 'link_remote_addr': 254, 'log_levels':
['NORMAL']}
2018-11-19 12:51:16,283 (dnp3agent-1.1 3029) dnp3.base_dnp3_agent
DEBUG: Loading DNP3 point definitions.
2018-11-19 12:51:16,283 (dnp3agent-1.1 3029) dnp3.points DEBUG:
Loaded 2 PointDefinitions
2018-11-19 12:51:17,543 (dnp3agent-1.1 3029) dnp3.outstation
DEBUG: DNP3Log TCPServer.cpp(118)     (filters=8) Accepted
connection from: 192.168.145.1
```

Figure 2. DNP3 Agent Logs



```
2018-11-19 13:34:45,019 (master_driveragent-3.1.1 7823) master_driver.driver DEBUG: publishing: devices/campus/building/all
2018-11-19 13:34:45,029 (master_driveragent-3.1.1 7823) master_driver.driver DEBUG: finish publishing: devices/campus/building/all
2018-11-19 13:34:45,032 (listeneragent-3.2 7843) listener.agent INFO: Peer: 'pubsub', Sender: 'platform.driver':, Bus: u'', Topic: 'devices/campus/building/all', Heade
rs: {'Date': '2018-11-19T21:34:45.019162+00:00', 'TimeStamp': '2018-11-19T21:34:45.019162+00:00', 'min_compatible_version': '3.0', 'max_compatible_version': u''}, Mess
age:
[{'REF615_1 Switch Status Open': '0',
  'REF615_1 Switch status Close': '0',
  'REF615_2 Switch Status Open': '0',
  'REF615_2 Switch status Close': '0',
  'REF615_3 Switch Status Open': '0',
  'REF615_3 Switch status Close': '0',
  'REF615_4 Switch Status Open': '0',
  'REF615_4 Switch status Close': '0',
  'REF615_5 Switch Status Open': '0',
  'REF615_5 Switch status Close': '0',
  'REF615_6 Switch Status Open': '0',
  'REF615_6 Switch status Close': '0',
  'REF615_7 Switch Status Open': '0',
  'REF615_7 Switch status Close': '0',
  'REF615_8 Switch Status Open': '0',
  'REF615_8 Switch status Close': '0'},
 {'REF615_1 Switch Status Open': {'type': 'integer',
                                  'tz': 'US/Pacific',
                                  'units': 'Point2'},
  'REF615_1 Switch status Close': {'type': 'integer',
                                   'tz': 'US/Pacific',
                                   'units': 'Point1'},
  'REF615_2 Switch Status Open': {'type': 'integer',
                                  'tz': 'US/Pacific',
                                  'units': 'Point4'},
  'REF615_2 Switch status Close': {'type': 'integer',
                                   'tz': 'US/Pacific',
```

Figure 3. DNP3 Data Points on the VOLTTRON™ Message Bus

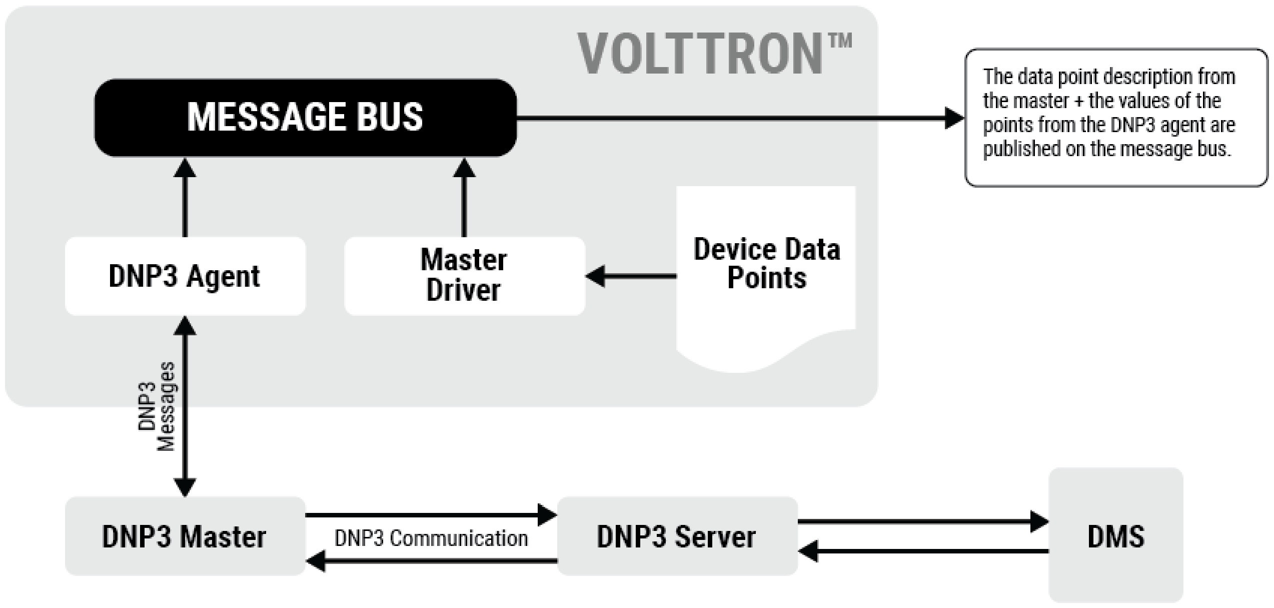

VOLTTRON™
MESSAGE BUS
The data point description from the master + the values of the points from the DNP3 agent are published on the message bus.
DNP3 Agent
Master Driver
Device Data Points
DNP3 Messages
DNP3 Master
DNP3 Communication
DNP3 Server
DMS

Figure 4. Overview of DNP3 Communication

## 2.2 Master Driver Processing

The master driver can communicate with any grid system based on its configuration. In the KISS use-cases, the driver communicates with the EMS/DMS to receive data from devices, such as an RTU, creates a docket by interfacing with LinkLite, and archives it in the historian.

### 2.2.1 Data Flow

The master driver requests docket creation for the raw message using the routing key ***gt.create_docket*** in the VOLTTRON™ exchange. The LinkLite service then receives a request on a queue bound to the ***gt.create_docket*** key and returns a docket as the response to the queue. The historian receives the docket for archiving from the master driver. Before archiving, the historian verifies the integrity and relevance of the docket by sending a request using the routing key ***gt.verify_docket*** in the VOLTTRON™ exchange. The LinkLite service then accepts the requests on a queue bound to the ***gt.verify_docket*** key and returns the result of the verification process. The preceding process is a means of verifying that the docket is signed and cryptographically sound using the KSI signature embedded in the docket.

The LinkLite service can also be configured with granular policy, providing data provenance verification, specific device requirements, and other detailed policy verification capabilities.

Upon testing KISS's capabilities to create and verify dockets through the GridAware dashboard (addressed in Section 3.0), the KISS team will define the policies[4] (policies may vary based on the application and user/customer requirements), requirements, and constraints that need to be programmed into the LinkLite service in order to ensure cyber secure data exchanges and verification processes. Upon customizing the LinkLite service policies, the KISS team will test the robustness of the system by injecting cyber-attacks such as man-in-the-middle, data spoofing, data loss/theft, historical data manipulation, stealth data injection, etc. The results from these attack tests will be presented in future documents. The objective of those scenarios is to test the KISS subsystem for its resilience against common and effective cyber-attack and to demonstrate the increased cybersecurity of the overall connected grid systems using KSI blockchain technology. Once the data are stored as a docket, they are now imbued with cryptographic immutability and they are highly portable. The consumers of these data (permissioned/registered users) can then request the signed and verified docket from the historian. At any point in time, permitted/registered users can then verify the received docket from the historian with the LinkLite service and specific policy. The KISS team is in the process of articulating and developing test cases (test cases are defined in section 5.0) in which the KISS system will be tested against cyber-attacks such as data spoofing, data injection, unauthorized user access, etc.  is a sequence diagram illustrating the data flow.

[4] KISS team is exploring policy definitions. Policy definition in blockchain is use-case/application specific and depends on user requirements. Herein the word "policies" will be coded into another software module called "Sentry", which is a module within the Guardtime KSI blockchain process. At any given point and for any operation, the KISS will check the operation request against the policies (a.k.a., "against what is required in order to be compliant for this operation to execute) defined in Sentry.

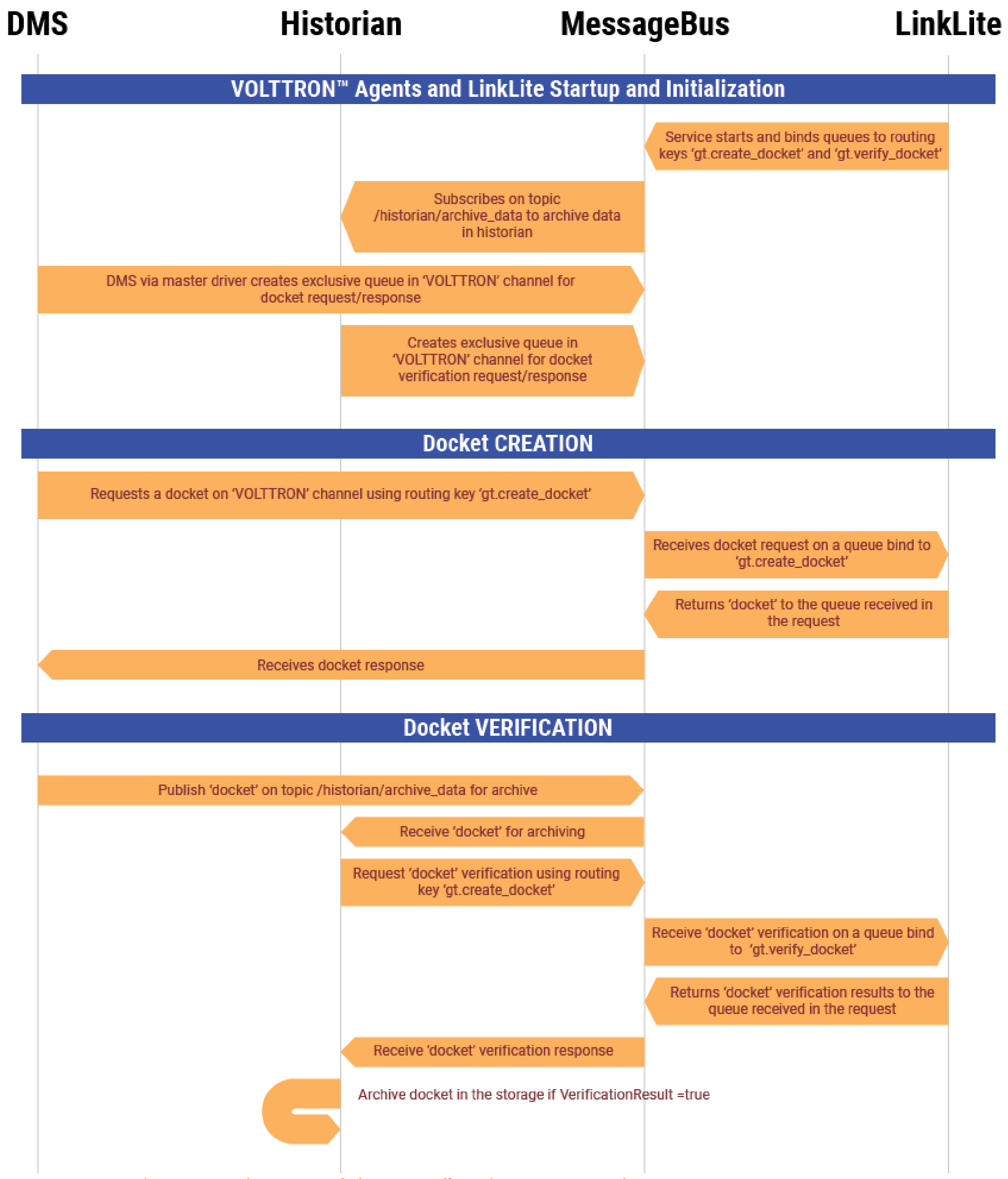

DMS
Historian
MessageBus
LinkLite
VOLTTRON™ Agents and LinkLite Startup and Initialization
Service starts and binds queues to routing keys 'gt.create_docket' and 'gt.verify_docket'
Subscribes on topic /historian/archive_data to archive data in historian
DMS via master driver creates exclusive queue in 'VOLTTRON' channel for docket request/response
Creates exclusive queue in 'VOLTTRON' channel for docket verification request/response
Docket CREATION
Requests a docket on 'VOLTTRON' channel using routing key 'gt.create_docket'
Receives docket request on a queue bind to 'gt.create_docket'
Returns 'docket' to the queue received in the request
Receives docket response
Docket VERIFICATION
Publish 'docket' on topic /historian/archive_data for archive
Receive 'docket' for archiving
Request 'docket' verification using routing key 'gt.create_docket'
Receive 'docket' verification on a queue bind to 'gt.verify_docket'
Returns 'docket' verification results to the queue received in the request
Receive 'docket' verification response
Archive docket in the storage if VerificationResult =true

Figure 5. Diagram of the Dataflow between Various KISS Components

## 3.0 KISS GridAware Dashboard

The GridAware dashboard provides multiple levels of data visualization to the users about the aggregated state of events and data associated with the KISS instance. Visualization is provided in the dashboard using two views: **GridAware Dashboard Rollup View** and **GridAware Dashboard Details View**. The GridAware dashboard is a frontend service with user accessible features that visualize all the background processes that happen between VOLTTRON$^{TM}$ and GridAware (a.k.a., KISS processes). These processes include docket creation, docket verification, and storage. By using the dashboard, the user is not required to have the skill set to interact with the software in its raw state. Instead, the user will be able to track and verify dockets through the GridAware dashboard.

The **Dashboard Rollup View** provides an aggregate view of various events, commands, and other impactful data captured over the period of time selected. The event types that are being visualized are dynamic based on the docket information and VOLTTRON™ configuration. The GridAware dashboard provides a dynamic platform for visualizing many types of data sets for specific purposes. The data acquired by the dashboard frontend include events and commands that are deemed necessary to visualize and monitor for security, audit, or functional purposes. Some examples of event types being captured and visualized are:

- RTU provisioning events
- DMS commands and events
- DMS data from unknown RTUs
- configuration or state change of RTUs.

The dashboard screen capture depicted in Figure 6 illustrates 82 DMS commands and events over the course of a short time period. The dashboard is in the early development stages; the final product will have clear delineations of the content visualized.

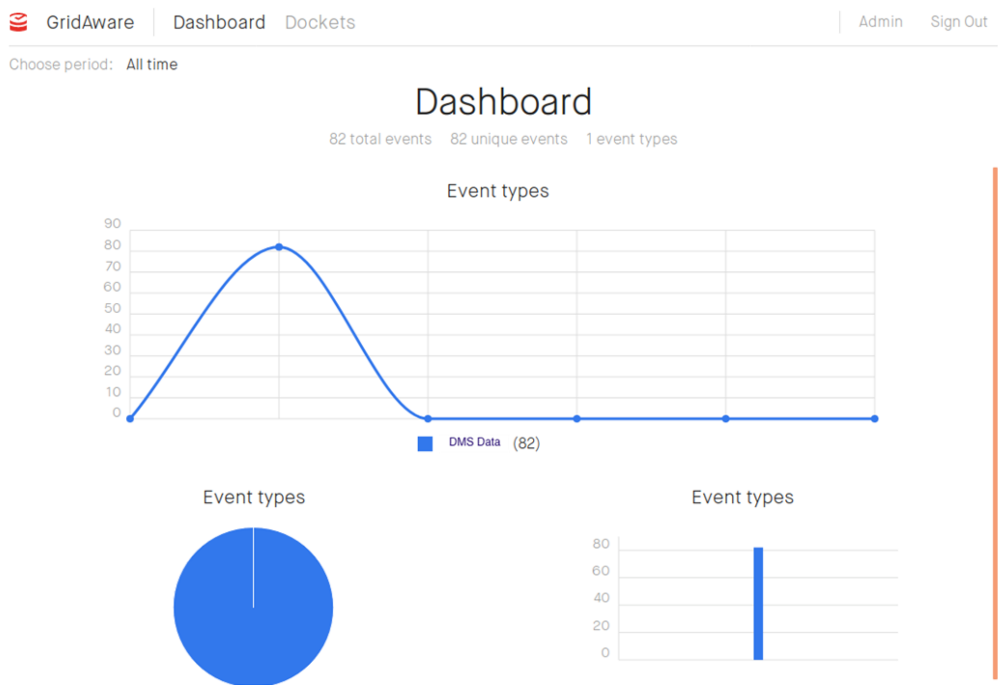
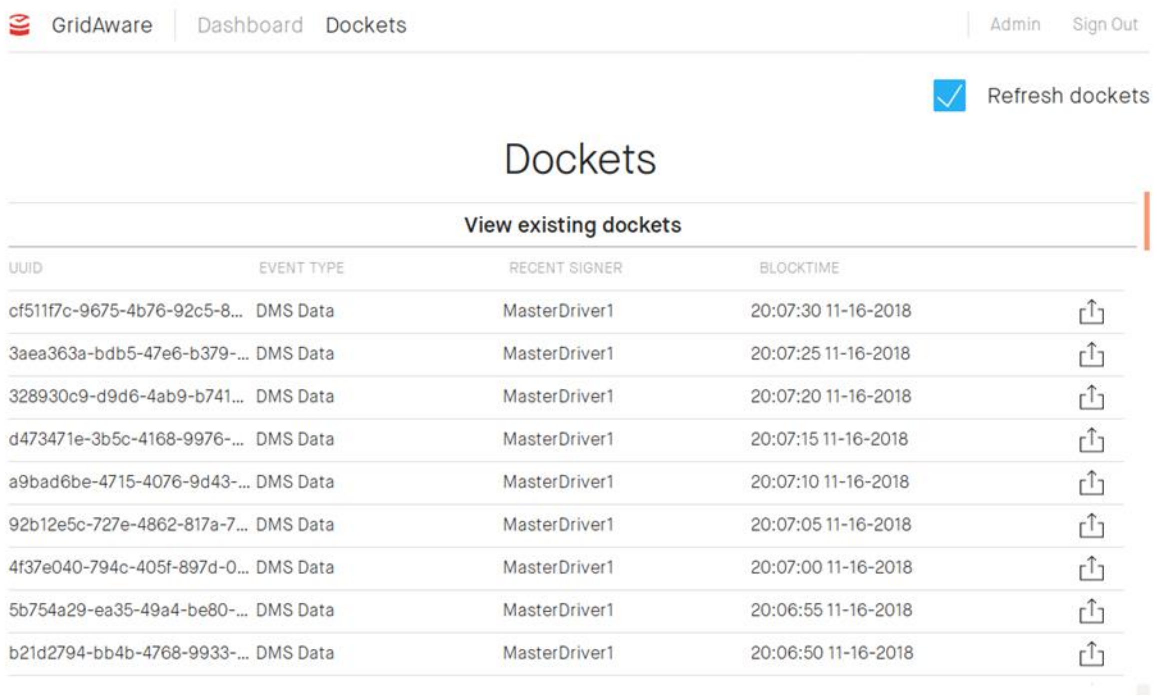


Figure 6. KISS GridAware Dashboard Rollup View Depicting Illustrative Events

The **GridAware Dashboard Details View** provides the users with the ability to select individual captured events and associated data about the event in the form of a docket. The details view provides the user with data useful for investigating details of captured data and events, and provides the ability to sort through the dockets with basic sorting functions. Figure 7 and Figure 8 are screen captures of example detailed views.

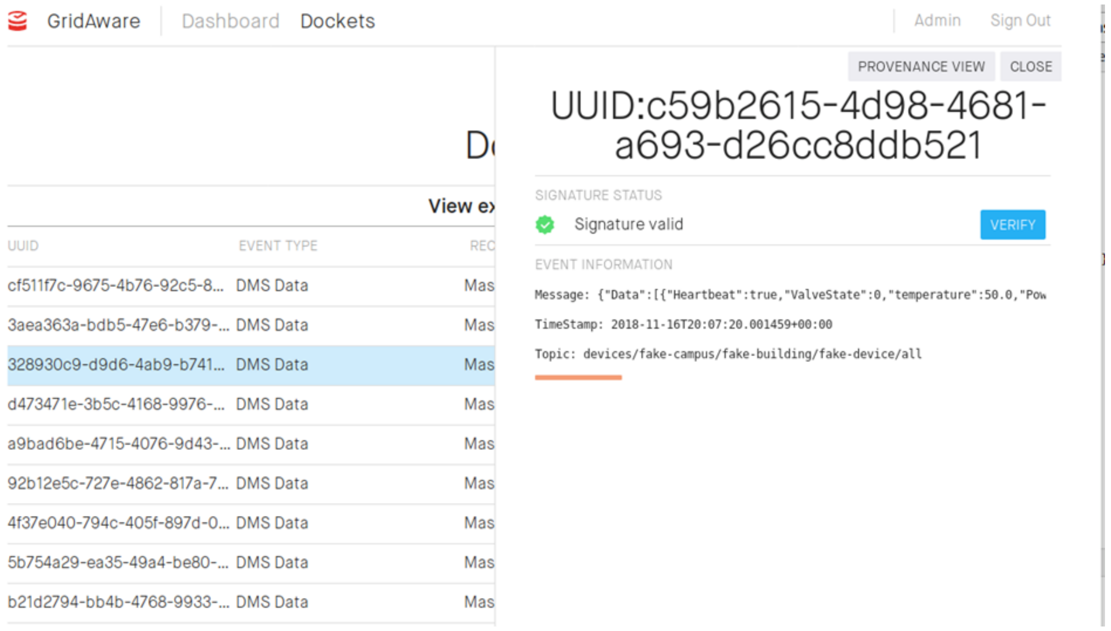


Dockets

View existing dockets

| UUID | EVENT TYPE | RECENT SIGNER | BLOCKTIME |
|---|---|---|---|
| cf511f7c-9675-4b76-92c5-8... | DMS Data | MasterDriver1 | 20:07:30 11-16-2018 |
| 3aea363a-bdb5-47e6-b379-... | DMS Data | MasterDriver1 | 20:07:25 11-16-2018 |
| 328930c9-d9d6-4ab9-b741... | DMS Data | MasterDriver1 | 20:07:20 11-16-2018 |
| d473471e-3b5c-4168-9976-... | DMS Data | MasterDriver1 | 20:07:15 11-16-2018 |
| a9bad6be-4715-4076-9d43-... | DMS Data | MasterDriver1 | 20:07:10 11-16-2018 |
| 92b12e5c-727e-4862-817a-7... | DMS Data | MasterDriver1 | 20:07:05 11-16-2018 |
| 4f37e040-794c-405f-897d-0... | DMS Data | MasterDriver1 | 20:07:00 11-16-2018 |
| 5b754a29-ea35-49a4-be80-... | DMS Data | MasterDriver1 | 20:06:55 11-16-2018 |
| b21d2794-bb4b-4768-9933-... | DMS Data | MasterDriver1 | 20:06:50 11-16-2018 |

Figure 7. KISS GridAware Dashboard Depicting the Existing Dockets, Associated Universally Unique Identifier (UUID), Event Type, Signer, and the Block Creation Time



| UUID | EVENT TYPE | REC |
|---|---|---|
| cf511f7c-9675-4b76-92c5-8... | DMS Data | Mas |
| 3aea363a-bdb5-47e6-b379-... | DMS Data | Mas |
| 328930c9-d9d6-4ab9-b741... | DMS Data | Mas |
| d473471e-3b5c-4168-9976-... | DMS Data | Mas |
| a9bad6be-4715-4076-9d43-... | DMS Data | Mas |
| 92b12e5c-727e-4862-817a-7... | DMS Data | Mas |
| 4f37e040-794c-405f-897d-0... | DMS Data | Mas |
| 5b754a29-ea35-49a4-be80-... | DMS Data | Mas |
| b21d2794-bb4b-4768-9933-... | DMS Data | Mas |

UUID:c59b2615-4d98-4681-a693-d26cc8ddb521

SIGNATURE STATUS

Signature valid

EVENT INFORMATION

```
Message: {"Data":[{"Heartbeat":true,"ValveState":0,"temperature":50.0,"Pow

TimeStamp: 2018-11-16T20:07:20.001459+00:00

Topic: devices/fake-campus/fake-building/fake-device/all
```

Figure 8. KISS GridAware Dashboard Depicting the Contents of a Docket Such as the Data Value and Timestamp

## 4.0 Interface Design

The LinkLite service interfaces with the VOLTTRON™ platform via RabbitMQ and leverages an RPC API Request/Response-styled message pattern for data publication and consumption. Each client creates an exclusive default queue and sends the default queue along with the request for LinkLite to send back the response. The docket is the basic construct of GridAware components and is used as the data format for messages within KISS components.

1. VOLTTRON™ RabbitMQ: VOLTTRON™ as a part of the platform creates a RabbitMQ environment with the following attributes, and it will be used by KISS interface:

    a. Request: Request creates an exclusive default queue and correlation identifier (ID), which are added to the properties field of the RabbitMQ publish API.

    b. Response: Response is published to a queue received in the request. Response is a serialized JavaScript Object Notation (JSON) format string.

2. Docket creation: The VOLTTRON™ agent requests LinkLite to create a docket by publishing the request to a VOLTTRON™ exchange using routing key *gt.create_docket***.**

    a. Request: The docket creation request uses a queue described below and an exchange defined by the VOLTTRON™ platform. The requests are JSON serialized strings that have headers and message fields.

    b. Response: The response is published to a queue received in the request. The response is a JSON serialized string with a docket appended to the UUID string object received in the request.

3. Docket verification: For verification of a docket, a VOLTTRON™ agent requests LinkLite to verify a docket for data integrity by publishing to a VOLTTRON™ exchange using the routing key *gt.verify_docket***.**

    a. Request: The docket creation request uses a queue described below and an exchange defined by the VOLTTRON™ platform. The requests are JSON serialized strings that have headers and message fields.

    b. Response: The response is published to a queue received in the request. A response is a JSON serialized string with a docket appended to the universally unique identifier (UUID) string object received in the request.

NOTE: The detailed sequence of events, syntax, and illustrations are shown in Appendix B.

## 5.0 Next Steps

The initial KISS design and development effort focused on specifically addressing integration of VOLTTRON™ and GridAware and showing data integrity through docket creation and verification (considered Phase 1 of development). The design phases demonstrated in this report are focused on the base functionality of KISS subsystem and does not address software aspects such as policy definitions in sentry, verification and validation process through KSI blockchain. The demonstrated design phases are for tool development and not considered the phases of the project. Specific areas that the KISS team will address in further development is additional data integrity using real-time data and data provenance.

- Phase 2: Develop a “player” that takes the data that has been harvested from the EMS/DMS and inputs it into a separate data base such as the MySQL or other location to enable verification of whether or not the docket has been manipulated.
- Phase 3: Integration with PNNL’s EMS and WSU’s DMS and documenting rules for data specific to those systems.
- Phase 4: Provenance of tool which includes access authorization and authentication to ensure the integrity of the data, confidentiality of the devices exchanging data with the KISS platform in the loop. The KISS team will ensure that the additional system and data integrity elements enforced by KISS will not add data leakages. This phase also includes addressing potential latency issues.
- Throughout the development cycle through completion, unit testing, integration testing, functional testing and independent tool testing will be performed. The testing will be performed by evaluating the KISS system response to attack scenarios. Some of the cyber-attacks that the KISS team is planning to develop may include:
    - Man-in-the-middle attack
    - Spoofing attacks such as data theft and data manipulation attack
    - Unauthorized user access to the data
    - Disruption of services
    - Other attacks and scenarios as determined by the KISS team in testing phases

# Appendix A – KISS Agent Setup

The following sections highlight the repositories, requirements, and instructions for implementing KISS in its existing development.

## A.1 PNNL Repositories

The following are the development repositories used for tool development:

- https://stash.pnnl.gov/projects/KISS
- kiss-based-agent (master branch)
- VOLTTRON-kiss (KISS-enabled branch [set as default]).

## A.2 Requirements

A current requirement during tool development is a xenial base computer (Ubuntu 16.04, mint 18.x).

## A.3 Instructions

The following instructions provide a brief introduction to agent installation in the VOLTTRON™ environment. This setup requires that at least three shells be open: (1) the first one will be for the VOLTTRON™ command and tailing of the volttron.log, (2) the second one will be for issuing installation commands to the VOLTTRON™ instance, and (3) the third shell will be for LinkLite. Based on the initial tests performed, it is recommended to execute the below commands sequentially in the order defined below.

### A.3.1 VOLTTRON™ Installation (Shell 1)

1. Clone both of the above repositories to your `$HOME` directory.
2. Open a shell to `$HOME/VOLTTRON™ ($VOLTTRON_ROOT`).
3. Install the rabbitmq dependencies by executing `bash $VOLTTRON_ROOT/rabbitmq-volttron/scripts/rabbit_dependencies.sh xenial.`
4. Bootstrap VOLTTRON™ by executing `python bootstrap.py –rabbitmq.`
5. Activate the environment by executing `source $VOLTTRON_ROOT/env/bin/activate.`
6. Install a master driver (not kiss version) `vcfg --agent master_driver.`
7. Install a listener agent `vcfg --agent listener.`
8. Start VOLTTRON™ by executing `$VOLTTRON_ROOT/start-volttron.sh.`
9. Tail the log by executing `tailf volttron.log.`

### A.3.2 KISS Agents (Shell 2)

1. Activate the shell by executing source `$VOLTTRON_ROOT/env/bin/activate.`
2. Change directory to kiss-based-agents `cd $HOME/kiss-based-agents` (`$KISS_AGENTS`).

3. Install KISS master driver agent by executing `bash $KISS_AGENTS/KISSMasterDriverAgent/install-agent.sh.`
4. Install KISS historian agent by executing `bash $KISS_AGENTS/KISSSQLHistorian/install-agent.sh.`

### A.3.3 LinkLite Installation (Shell 3)

1. Install `Java JDK 1.8` or greater.
2. Install `python-devel` package.
3. Install `LinkLite` debian package.
    a. Run dbpg-deb -x link lite.deb <install directory path>
        - e.g., dbpg-deb -x linklite.deb /home/user/tmp/
        - LinkLite package is installed in `linklite` sub-directory from the input install directory.
            – e.g., /home/user/tmp/linklite/.
4. Configure KSI, RabbitMQ server and login credentials in `linklite/config/application.properties` file.

```
ksi.agr.userid=<aggregator user>
ksi.agr.secret=<aggregator key>
ksi.ext.userid=<extender user>
ksi.ext.secret=<extender key>
rmq.host = <RMQ server>
rmq.port = 5671
rmq.user = <RMQ username>
rmq.pwd = <RMQ password>
rmq.vhost = <volttron>
rmq.exchange = <volttron>
```

5. Start RabbitMQ and VOLTTRON™ 6 platform if is not running.
6. Start `LinkLite` Service `unkg init` script, which will run as a background process detached from the terminal.
    - linklite/init.d/linklite
    - Logs are located in the directory linklite/log/linklite.log.

# Appendix B – KISS Sequence of Events: RabbitMQ, Docket Creation, and Docket Verification

## B.1 VOLTTRON™ RabbitMQ

VOLTTRON™, as a part of a platform, creates RabbitMQ environment with the following attributes and will be used by KISS interface:

```
Virtual Host:volttron
Exchange:volttron
```

### B.1.1 Request

Request creates an exclusive default queue and a correlation ID, which are added to the properties field of the RabbitMQ publish API. It uses the following JSON format:

```
{
  "Data":{
    "UUID":"uuid value",
    "Payload":{
      "Headers":{
        "TimeStamp":"TimeStamp",
        "AgentId":"Agentid"
      },
      "Message":"agent data"
    }
  }
}
```

where,

UUID contains UniqueId.
Headers contains agent metadata that includes TimeStamp and AgentId
Message contains agent data, which can be simple JSON String value or a JSON object or a JSON array.

### B.1.2 Response

Response is published to a queue received in the request. Response is a serialized JSON format string.

## B.2 API

### B.2.1 Docket Creation

The VOLTTRON™ agent requests LinkLite to create a docket by publishing the request to a *VOLTTRON™* exchange using routing key *gt.create_docket*.

### B.2.2 Request

The docket creation request uses a queue described below and an exchange defined by the VOLTTRON™ platform. The requests are JSON serialized strings that have headers and message fields.

exchange = volttron
routing key = gt.create_docket
properties.reply_to = <queue to send the response>

*{"Data": {"UUID"::uuid value", "Payload":{"Headers": { "TimeStamp": "TimeStamp", "AgentId": "agentid"}, "Message": "agent data for which docket requested"}}*

where,
Headers field has agent metadata.
Message field has agent data for which docket is requested, it can be simple string value or a JSON object or a JSON array.

**sample request message:**
{"Data": {"UUID": "6d3dd987-7c51-440e-9704-f2835760e8d3", "Payload": {"Headers": {"TimeStamp": "2018-45-10 16:45:28", "id": "volttron_agent"}, "Message": {"data": ["value1", "value2"], "data2": ["value1"]}}}}

### B.2.3 Response

The response is published to a queue received in the request. The response is a JSON serialized string with a docket appended to the UUID string object received in the request.

*{"UUID":"uuid value received in the request" ,"Docket":"XML string","AgentId":"agentid"}*

**sample response message:**
{"UIID": "96c08581-f70b-4dec-9cf3-f960cbcfed4f", "AgentId": "gt.create_docket", "Docket": "<?xml version=\"1.0\" encoding=\"UTF-8\" standalone=\"no\"?><Docket signatureContentType=\"1\" version=\"kiss-1.0\"><Data>96c08581-f70b-4dec-9cf3-f960cbcfed4f</Data><Properties><Payload>{\"Message\":{\"data\":[\"value1\",\"value2\"],\"data2\":[\"value1\"]},\"Headers\":{\"id\":\"volttron_agent\",\"TimeStamp\":\"2018-50-12 16:50:34\"}}</Payload></Properties><KSig>iAAHrogBAHICBFvBM0sDAQ8DAgMFAwE=</KSig></Docket>"}

### B.2.4 Sequence of Events

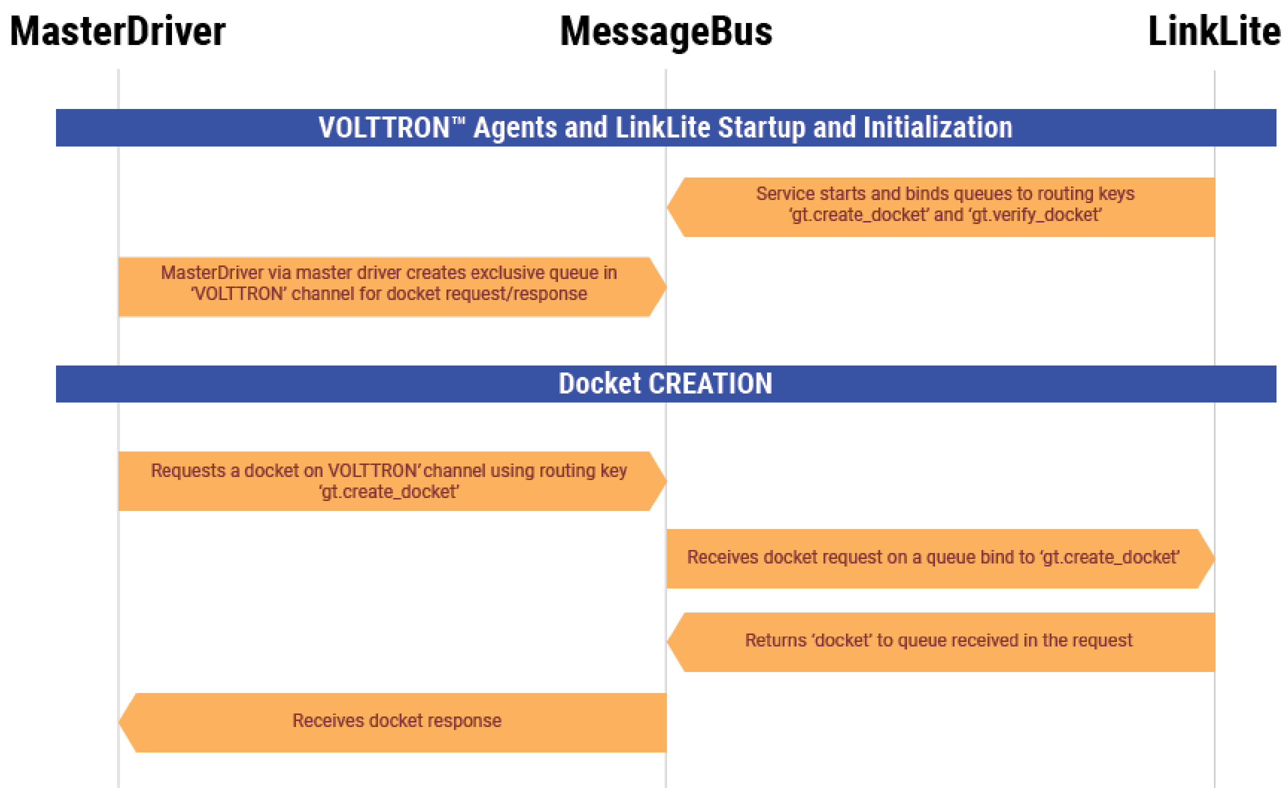

MasterDriver
MessageBus
LinkLite
VOLTTRON™ Agents and LinkLite Startup and Initialization
Service starts and binds queues to routing keys 'gt.create_docket' and 'gt.verify_docket'
MasterDriver via master driver creates exclusive queue in 'VOLTTRON' channel for docket request/response
Docket CREATION
Requests a docket on VOLTTRON' channel using routing key 'gt.create_docket'
Receives docket request on a queue bind to 'gt.create_docket'
Returns 'docket' to queue received in the request
Receives docket response

Figure B.1. Docket Creation Sequence of Events

## B.3 Docket Verification

For verification of a docket, a VOLTTRON™ agent requests LinkLite to verify a docket for data integrity by publishing to a VOLTTRON™ exchange using the routing key *gt.verify_docket*.

### B.3.1 Request

The docket creation request uses a queue described below and an exchange defined by the VOLTTRON™ platform. The requests are JSON serialized strings that have headers and message fields.

exchange = VOLTTRON™
routing key = gt.verify_docket
properties.reply_to = <queue to send the response>

*{"Data":{"UUID": "uuid value", "Payload": {"Headers": {"TimeStamp": "Message time stamp", "AgentId": "historian_agent"}, "Message": {"Docket" :"docket string"}}}}*

where,
  Headers field has metadata.
  Message field has a Docket String for which verification is requested.

**sample request message:**

{"Data": {"UUID": "ab4f6d17-f8ef-4db4-a96f-3ae0256d25df", "Payload": {"Headers": {"TimeStamp": "2018-50-12 16:50:39", "id": "historian_agent"}, "Message": {"Docket": "<?xml version=\"1.0\" encoding=\"UTF-8\" standalone=\"no\"?><Docket signatureContentType=\"1\" version=\"kiss-1.0\"><Data>96c08581-f70b-4dec-9cf3-f960cbcfed4f</Data><Properties><Payload>{\"Message\":{\"data\":[\"value1\",\"value2\"],\"data2\":[\"value1\"]},\"Headers\":{\"id\":\"volttron_agent\",\"TimeStamp\":\"2018-50-12 16:50:34\"}}</Payload></Properties><KSig>iAAHrogBAHICBFvBM0sDAQ8DAgMFAwE=</KSig></Docket>"}}}}

### B.3.2 Response

The response is published to a queue received in the request. A response is a JSON serialized string with a docket appended to the UUID string object received in the request.

*{UUID":"uuid value received in the request",,"VerficationResult":"true or false","AgentId":"agentid"}*

**sample response message:**

{"VerificationResult":"true","UIID":"ab4f6d17-f8ef-4db4-a96f-3ae0256d25df","AgentId":"gt.verify_docket"}

### B.3.3 Sequence of Events

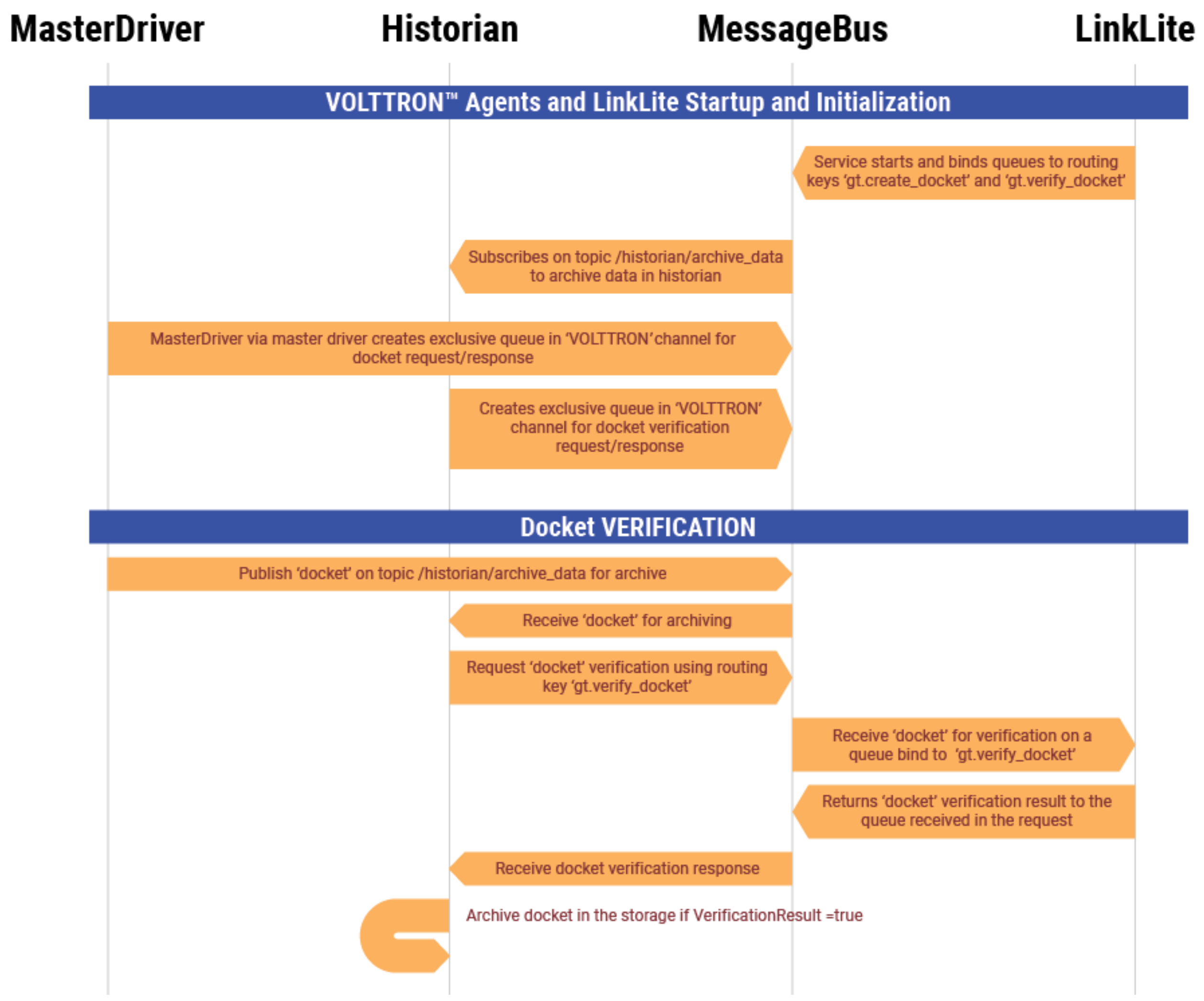

MasterDriver
Historian
MessageBus
LinkLite
VOLTTRON™ Agents and LinkLite Startup and Initialization
Service starts and binds queues to routing keys 'gt.create_docket' and 'gt.verify_docket'
Subscribes on topic /historian/archive_data to archive data in historian
MasterDriver via master driver creates exclusive queue in 'VOLTTRON' channel for docket request/response
Creates exclusive queue in 'VOLTTRON' channel for docket verification request/response
Docket VERIFICATION
Publish 'docket' on topic /historian/archive_data for archive
Receive 'docket' for archiving
Request 'docket' verification using routing key 'gt.verify_docket'
Receive 'docket' for verification on a queue bind to 'gt.verify_docket'
Returns 'docket' verification result to the queue received in the request
Receive docket verification response
Archive docket in the storage if VerificationResult =true

Figure B.2. Docket Verification Sequence of Events