THALES

Thales Luna Network HSM 7

SDK REFERENCE

Document Information

Last Updated	2026-05-28 09:26:37 GMT-05:00

Disclaimer
All information herein is either public information or is the property of and owned solely by Thales Group and/or its subsidiaries who shall have and keep the sole right to file patent applications or any other kind of intellectual property protection in connection with such information.

Nothing herein shall be construed as implying or granting to you any rights, by license, grant or otherwise, under any intellectual and/or industrial property rights of or concerning any of Thales Group's information.

This document can be used for informational, non-commercial, internal, and personal use only provided that:

> The copyright notice, the confidentiality and proprietary legend and this full warning notice appear in all copies.

> This document shall not be posted on any publicly accessible network computer or broadcast in any media, and no modification of any part of this document shall be made.

Use for any other purpose is expressly prohibited and may result in severe civil and criminal liabilities.

The information contained in this document is provided "AS IS" without any warranty of any kind. Unless otherwise expressly agreed in writing, Thales Group makes no warranty as to the value or accuracy of information contained herein.

The document could include technical inaccuracies or typographical errors. Changes are periodically added to the information herein. Furthermore, Thales Group reserves the right to make any change or improvement in the specifications data, information, and the like described herein, at any time.

Thales Group hereby disclaims all warranties and conditions with regard to the information contained herein, including all implied warranties of merchantability, fitness for a particular purpose, title and non-infringement. In no event shall Thales Group be liable, whether in contract, tort or otherwise, for any indirect, special or consequential damages or any damages whatsoever including but not limited to damages resulting from loss of use, data, profits, revenues, or customers, arising out of or in connection with the use or performance of information contained in this document.

Thales Group does not and shall not warrant that this product will be resistant to all possible attacks and shall not incur, and disclaims, any liability in this respect. Even if each product is compliant with current security standards in force on the date of their design, security mechanisms' resistance necessarily evolves according to the state of the art in security and notably under the emergence of new attacks. Under no circumstances, shall Thales Group be held liable for any third party actions and in particular in case of any successful attack against systems or equipment incorporating Thales products. Thales Group disclaims any liability with respect to security for direct, indirect, incidental or consequential damages that result from any use of its products. It is further stressed

that independent testing and verification by the person using the product is particularly encouraged, especially in any application in which defective, incorrect or insecure functioning could result in damage to persons or property, denial of service, or loss of privacy.

Regulatory Compliance

This product complies with the following regulatory regulations. To ensure compliancy, ensure that you install the products as specified in the installation instructions and use only Thales-supplied or approved accessories.

USA, FCC

This equipment has been tested and found to comply with the limits for a “Class B” digital device, pursuant to part 15 of the FCC rules.

Canada

This class B digital apparatus meets all requirements of the Canadian interference-causing equipment regulations.

Europe

This product is in conformity with the protection requirements of EC Council Directive 2014/30/EU. This product satisfies the CLASS B limits of EN55032.

CONTENTS

PREFACE: About the REST API Guide

Appliance administrators now have the ability to use a representational state transfer application programming interface—REST-ful API—to configure and query the Luna Network HSM, in addition to the long-standing Luna shell (LunaSH).

REST API provides a lightweight architecture that allows communication between HSM appliances and your own web applications.

This guide contains the following information and procedures:

The online version of this documentation also includes reference material for each released version of the Luna REST API.

The preface includes the following information about this document:

For information regarding the document status and revision history, see "Document Information" on page 2.

Customer Release Notes

The Customer Release Notes (CRN) provide important information about specific releases. Read the CRN to fully understand the capabilities, limitations, and known issues for each release. You can view the latest version of the CRN at www.thalesdocs.com.

Audience

This document is intended for personnel responsible for maintaining your organization's security infrastructure. This includes Luna HSM users and security officers, key manager administrators, and network administrators.

All products manufactured and distributed by Thales are designed to be installed, operated, and maintained by personnel who have the knowledge, training, and qualifications required to safely perform the tasks assigned to them. The information, processes, and procedures contained in this document are intended for use by trained and qualified personnel only.

It is assumed that the users of this document are proficient with security concepts.

Document Conventions

This document uses standard conventions for describing the user interface and for alerting you to important information.

Notes

Notes are used to alert you to important or helpful information. They use the following format:

> **NOTE** Take note. Contains important or helpful information.

Cautions

Cautions are used to alert you to important information that may help prevent unexpected results or data loss. They use the following format:

> **CAUTION!** Exercise caution. Contains important information that may help prevent unexpected results or data loss.

Warnings

Warnings are used to alert you to the potential for catastrophic data loss or personal injury. They use the following format:

> ****WARNING** Be extremely careful and obey all safety and security measures. In this situation you might do something that could result in catastrophic data loss or personal injury.**

Command syntax and typeface conventions

Format	Convention
bold	The bold attribute is used to indicate the following: > Command-line commands and options (Type **dir /p**.) > Button names (Click **Save As**.) > Check box and radio button names (Select the **Print Duplex** check box.) > Dialog box titles (On the **Protect Document** dialog box, click **Yes**.) > Field names (**User Name**: Enter the name of the user.) > Menu names (On the **File** menu, click **Save**.) (Click **Menu** > **Go To** > **Folders**.) > User input (In the **Date** box, type **April 1**.)
italics	In type, the italic attribute is used for emphasis or to indicate a related document. (See the *Installation Guide* for more information.)
<variable>	In command descriptions, angle brackets represent variables. You must substitute a value for command line arguments that are enclosed in angle brackets.
[**optional**] [<optional>]	Represent optional **keywords** or <variables> in a command line description. Optionally enter the keyword or <variable> that is enclosed in square brackets, if it is necessary or desirable to complete the task.
{**a**|**b**|**c**} {<a>|<b>|<c>}	Represent required alternate **keywords** or <variables> in a command line description. You must choose one command line argument enclosed within the braces. Choices are separated by vertical (OR) bars.
[**a**|**b**|**c**] [<a>|<b>|<c>]	Represent optional alternate keywords or variables in a command line description. Choose one command line argument enclosed within the braces, if desired. Choices are separated by vertical (OR) bars.

Support Contacts

If you encounter a problem while installing, registering, or operating this product, please refer to the documentation before contacting support. If you cannot resolve the issue, contact your supplier or Thales Customer Support. Thales Customer Support operates 24 hours a day, 7 days a week. Your level of access is governed by the support plan negotiated between Thales and your organization. Please consult this plan for details regarding your entitlements, including the hours when telephone support is available to you.

Customer Support Portal

The Customer Support Portal, at https://supportportal.thalesgroup.com, is where you can find solutions for most common problems and create and manage support cases. It offers a comprehensive, fully searchable database of support resources, including software and firmware downloads, release notes listing known problems and workarounds, a knowledge base, FAQs, product documentation, technical notes, and more.

> **NOTE** You require an account to access the Customer Support Portal. To create a new account, go to the portal and click on the **REGISTER** link.

Telephone

The support portal also lists telephone numbers for voice contact (Contact Us).

CHAPTER 1: Webserver Setup

The REST API webserver comes installed on Luna Network HSMs. The appliance **admin** must first enable the webserver and configure it to accept commands from your web applications. Next, use a combination of REST API resources and Luna HSM Client tools to create an NTLS or STC client connection to a partition on the HSM.

Enabling the Webserver

1. Log in to LunaSH on the Luna Network HSM appliance as **admin**.
2. [Optional] Specify a network device for REST API traffic.

   lunash:> **webserver bind -netdevice** <netdevice> [**-port** <portnum>]
3. Generate a certificate for the webserver.

   lunash:> **webserver certificate generate -keytype rsa**

   > **NOTE** It is recommended that you use the RSA algorithm for this cryptographic operation.
4. Enable the webserver.

   lunash:> **webserver enable**
5. If you plan to issue REST commands via a browser using AJAX calls, set the permitted origin domains for the webserver by specifying a comma-separated list of approved REST API client domains, IPs, or IP/port combinations.

   lunash:> **webserver origin set -list** "<domain/IP_list>"
6. [Optional] If required, configure the webserver cipher suite by specifying a colon-separated list of ciphers (or all to enable all available ciphers).

   lunash:> **webserver ciphers set -list** <cipher_list>
7. Restart the webserver service.

   lunash:> **service restart webserver**

   > **NOTE** It will perform synchronization to remove discrepancies between REST API and LunaSH users/roles.
8. [Optional] Check the REST API status on the appliance.

   lunash:> **webserver show**

You can now issue REST API commands to the Luna Network HSM. See "HSM and Partition Setup" on page 4 to continue setting up the HSM and partitions.

CHAPTER 2: Authentication

This section contains the following information about authenticating to the webserver on the Luna Network HSM appliance using REST API:

> Sessions

> Basic Authorization

> Certificate-Based Authorization

Sessions

REST API sessions store valuable information required to use the service. Two types of sessions are available:

User Sessions: This is the default session type, used for basic authorization. User sessions do not require a session header. They are shared with all clients of the same user and are, therefore, granted limited access to resources.

Private Sessions: These are created using a resource (see POST /auth/session). Private sessions can only be used by the user who created them.

Basic Authorization

To use basic authorization, you must include the following in the **Authorization** header:

> keyword: **Basic**

> base64-encoded blob containing your colon-separated appliance username and password

Example: base64(admin:password123)

```
Authorization: Basic YWRtaW46cGFzc3dvcmQxMjM
```

Certificate-Based Authorization

Certificate-based authorization requires you to upload a user certificate and perform a login handshake.

1. Log in to the appliance with your username and password.
2. Upload the certificate for the specified user. Replace newline characters within the certificate string.

```
POST
https://LUNAIPADDRESS:PORT/users/{userid}/certificates
{"certificate": "-----BEGIN CERTIFICATE-----\n...{certificate}...\n-----END CERTIFICATE-----
\n"}
```

3. Generate a challenge for the user by specifying the username and client public certificate.

```
POST
https://LUNAIPADDRESS:PORT/auth/login/challenge
{
  "username": "{username}",
  "certificate": "-----BEGIN CERTIFICATE-----\n...{certificate}...\n-----END CERTIFICATE----
-\n"
}
```

The webserver responds with a base64-encoded cryptographic challenge and nonce parameters.

4. Decode the challenge and nonce from base64.
5. Decrypt the decoded challenge using the client private key.
6. XOR the decoded and decrypted challenge with the decoded nonce. The result is the plaintext challenge answer.
7. Encrypt the answer with the webserver's public key. The result is the challenge response expected by the server.
8. Encode the challenge response to base64 so that it can be transmitted via REST API.
9. Answer the challenge with the encrypted base64 response.

```
POST
https://LUNAIPADDRESS:PORT/auth/login/basic
{"challengeResponse": "{challenge_response}"}
```


CHAPTER 3: HSM and Partition Setup

The following procedures will allow you to initialize the HSM, set policies, create an application partition, and connect the partition to a client. You must first set up and enable the webserver on the appliance (see "Webserver Setup" on page 1).

- Initializing the HSM and Creating a Partition
- Creating an NTLS Client-Partition Connection
- Creating an STC Client-Partition Connection
- Converting an Initialized NTLS Partition-Client Connection to STC

REST API resources are used in place of LunaSH commands. Some REST API resources combine multiple LunaSH commands for efficiency. Therefore, many are not 1:1 mappings. The tables linked below provide a cross-reference for key resources used in the following procedures:

- **clientLushXRefPageName**
- **hsmLushXRefPageName**
- **ntlsLushXRefPageName**
- **partitionLushXRefPageName**
- **serviceLushXRefPageName**
- **stcLushXRefPageName**

NOTE

- Resources and objects are case-sensitive; use this documentation as a reference for the correct case.
- The default session timeout is 10 minutes; upon timeout, the REST server terminates an authenticated session. To keep the REST session alive, periodically query a resource such as GET /api/lunasa/webServer or GET /api/lunasa/syslog/logs/{logid} with a suitable log message (to create a record of the keep-alive action).
- HSM and partition serial numbers are unique; HSM and partition labels can be identical. Keep this point in mind when constructing logic for REST resources.
- If you encounter the string NO ERR WITH THIS ID EXISTS, please contact Thales Technical Support and report how you encountered it.
- LunaSH imposes constraints on names, labels, etc. to prevent the use of characters that could be used to exploit the shell. You should use the same character set with the REST API. Refer to the product documentation for more details.
- All discoverable resources (resources with a GET) will be referenced by an href in its parent resource; this documentation may not reflect all resources that follow this rule.

Initializing the HSM and Creating a Partition

1. Open a session on the Luna Network HSM appliance.

```
POST
https://LUNAIPADDRESS:PORT/auth/session
Authorization: Basic YWRtaW46MXFAVzNlJFI=
{}
```

2. Change any desired HSM policies.

```
PUT
https://LUNAIPADDRESS:PORT/api/lunasa/hsms/{hsmid}/policies/{policynum}
{"value":{value}}
```

3. Initialize the HSM.

```
PUT
https://LUNAIPADDRESS:PORT/api/lunasa/hsms/{hsmid}
{
  "label": "{HSMlabel}",
  "password": "{SO_password}",
  "defaultDomain": false,
  "domain": "{domain_string}"
}
```

4. Log in as HSM SO.

```
POST
https://LUNAIPADDRESS:PORT/api/lunasa/hsms/{hsmSN}/login
{
  "ped": "0",
  "password": "password",
  "role": "so"
}
```

5. [Optional] Run a self-test.

```
POST
https://LUNAIPADDRESS:PORT/api/lunasa/hsms/{hsmid}/actions/selfTest
```

6. Create a partition on the HSM.

```
POST
https://LUNAIPADDRESS:PORT/api/lunasa/hsms/{hsmid}/partitions
{
  "defaultDomain": true,
  "defaultChallenge": true,
  "name": "123",
  "allStorageSpace": false,
  "label": "123",
  "domain": "test",
  "hasPso": true,
  "password": "password123",
  "size": 20480
}
```

Next, allow a client to access the partition using an NTLS or STC connection.

Creating an NTLS Client-Partition Connection

1. [Optional] Get a list of clients currently registered on the appliance.


```
GET
https://LUNAIPADDRESS:PORT/api/lunasa/ntls/clients
```

2. Register the client with the appliance by specifying the client certificate as a string. Within the string, you must replace all newline characters with "\n".

```
POST
https://LUNAIPADDRESS:PORT/api/lunasa/ntls/clients
{
  "ip": "1.2.3.4",
  "certificate": "-----BEGIN CERTIFICATE-----\n...{certificate}...\n-----END CERTIFICATE----
-",
  "clientName": "testClient3"
}
```

3. Restart the NTLS service.

```
POST
https://LUNAIPADDRESS:PORT/api/lunasa/services/ntls/actions/restart
```

4. Assign the partition to the client.

```
POST
https://LUNAIPADDRESS:PORT/api/lunasa/ntls/clients/{clientid}/links
{"url": "/api/lunasa/hsms/{hsmid}/partitions/{partitionid}"}
```

5. [Optional] Get a list of partitions assigned to the client.

```
GET
https://LUNAIPADDRESS:PORT/api/lunasa/ntls/clients/{clientid}/links/{linkid}
{
  "url": "/api/lunasa/hsms/{hsmid}/partitions/{partitionid}",
  "type": "hsm/partition"
}
```

Creating an STC Client-Partition Connection

To create a Secure Trusted Channel (STC) connection, a partition identity is created directly on the partition, and the client and partition exchange identities. This allows end-to-end encryption of all communications between partition and client. This section describes how to establish an STC connection between a client and a new partition and initialize the partition over STC.

If you have the partition already initialized, you may follow the procedure mentioned in Converting an Initialized NTLS Partition-Client Connection to STC to create STC connection between partition and client.

1. On the client workstation, import the HSM Appliance Server Certificate (server.pem) from the appliance and register the HSM Server Certificate with the client.

   scp admin<host/IP>:server.pem <target_filename>

   vtl addServer -n <Network_HSM_hostname/IP> -c <server_certificate>

2. Launch LunaCM on the Luna HSM Client and create the STC token and client identity.

   lunacm:> **stc tokeninit -label** <token_label>

   lunacm:> **stc identitycreate -label** <ID_label>

3. Encode the identity file using **openssl** so it can be registered via REST API call.

   openssl base64 -in <ID_filename> **-out** <encode_ID_filename>

   You must remove all newline characters within the encoded file.

4. Log in as HSM SO.

```
POST
https://LUNAIPADDRESS:PORT/api/lunasa/hsms/{hsmid}/login
{
  "ped": "0",
  "password": "password",
  "role": "so"
}
```

5. Export the partition identity.

```
GET
https://LUNAIPADDRESS:PORT/api/lunasa/hsms/{hsmid}/partitions/{partitionid}/stc
```

6. Register the partition identity to the client using LunaCM. The fingerpint obtained is base64-encoded, so you must decode it first with openssl.

   Before decoding, create a file containing the base64-encoded fingerpint and remove all the newline characters (i.e. '\ n'). Then, move the content enclosed within these to the newline in order to maintain the required format.

   openssl base64 -d -in <encoded_parID_filename> **-out** <parID_filename>

   lunacm:> **stc partitionregister -file** <parID_filename>

7. Enable STC using LunaCM on the client.

   lunacm:> **clientconfig listservers**

   lunacm:> **stc enable -id** <server_ID>

   LunaCM restarts. If successful, the partition appears in the list of available slots.

8. Initialize the partition.

```
PUT
https://LUNAIPADDRESS:PORT/api/lunasa/hsms/{hsmid}/partitions/{partitionid}
{
  "password": "password",
  "label": "ABC123",
  "domain": "domain",
  "defaultDomain": true,
  "ped": "0"
}
```

9. Log in as partition SO.

```
POST
https://LUNAIPADDRESS:PORT/api/lunasa/hsms/{hsmid}/partitions/{partitionid}/login
{
  "ped": "0",
  "role": "so",
  "password":"password"
}
```

10. [Optional] Set partition policy 37, Enable Secure Trusted Channel, to 1 (ON) to reserve the partition for the creation of STC connections (not NTLS) with other clients.

```
PUT
https://LUNAIPADDRESS:PORT/api/lunasa/hsms/{hsmid}/partitions/{partitionid}/policies/37
{"value": 1}
```

11. Register the client identity with the HSM by specifying the encoded file content.


```
POST
https://LUNAIPADDRESS:PORT/api/lunasa/hsms/{hsmid}/partitions/{partitionid}/stc/clients
{
  "identity": "{encoded_ID_file_content}",
  "label": "{client_label}"
}
```

Converting an Initialized NTLS Partition-Client Connection to STC

If you have initialized partitions already assigned to a client using NTLS, you can use the following procedure to switch to a more secure STC connection. All of the client's assigned partitions on the specified Luna Network HSM must be converted.

> **NOTE** Before moving to the following procedure, please ensure that your partition is already initialized.

1. Launch LunaCM on the Luna HSM Client and create the STC token and client identity.

   lunacm:> **stc tokeninit -label** <token_label>

   lunacm:> **stc identitycreate -label** <ID_label>

   The STC client identity public key is automatically exported to:

   <client_install_directory>/data/client_identities/>

   > **NOTE** This step is not required if you have already created a client token and identity. Verify using **stc identityshow**. If you recreate the client identity, you will have to re-register any existing STC partitions.

2. Encode the identity file using **openssl** so it can be registered via REST API call.

   openssl base64 -in <ID_filename> **-out** <encode_ID_filename>

   You must remove all newline characters within the encoded file.

3. Log in as partition SO.

   ```
   POST
   https://LUNAIPADDRESS:PORT/api/lunasa/hsms/{hsmid}/partitions/{partitionid}/login
   {
     "ped": "0",
     "role": "so",
     "password":"password"
   }
   ```

4. [Optional] Set partition policy 37, Enable Secure Trusted Channel, to 1 (ON) to reserve the partition for the creation of STC connections (not NTLS) with other clients.

   ```
   PUT
   https://LUNAIPADDRESS:PORT/api/lunasa/hsms/{hsmid}/partitions/{partitionid}/policies/37
   {"value": 1}
   ```

5. Export the partition identity.

   ```
   GET
   https://LUNAIPADDRESS:PORT/api/lunasa/hsms/{hsmid}/partitions/{partitionid}/stc
   ```

6. Register the partition identity to the client using LunaCM. The fingerpint obtained is base64-encoded, so you must decode it first with **openssl**.

   Before decoding, create a file containing the base64-encoded fingerpint and remove all the newline characters (i.e. '\ n'). Then, move the content enclosed within these to the newline in order to maintain the required format.

   openssl base64 -d -in <encoded_parID_filename> **-out** <parID_filename>

   lunacm:> **stc partitionregister -file** <parID_filename>

7. Register the client identity with the HSM by specifying the encoded file content.

```
POST
https://LUNAIPADDRESS:PORT/api/lunasa/hsms/{hsmid}/partitions/{partitionid}/stc/clients
{
  "identity": "{encoded_ID_file_content}",
  "label": "{client_label}"
}
```

8. Enable STC using LunaCM on the client.

   lunacm:> **clientconfig listservers**

   lunacm:> **stc enable -id** <server_ID>

CHAPTER 4: Indirect Login

The indirect login capability allows you to use one Luna Network HSM to provide login credentials for a group of others. This can be useful when you need to configure multiple HSMs. The instructions below will allow you to configure indirect login.

As per the latest changes made in firmware 7.7.0, indirect login can be used in one of the following ways:

> Indirect Login on fw 7.4-and-older partitions

> Indirect Login on fw 7.7 partitions using key

> Indirect Login on fw 7.7 partitions using pkc

Indirect Login on FW 7.4-and-Older Partitions

In the examples below, **adminHSMid** refers to the serial number of the HSM (with appliance/firmware 7.7.0 and above) which contains the application partition (v0 or v1) that holds the private key used for indirect login, and **serviceHSMid** refers to the serial number of the HSM (with appliance/firmware 7.4.0 and below) which contains the administrator partition being configured.

Setting Up Indirect Login

1. Log in to an application partition on adminHSMid as the crypto officer (CO).
2. Export the public key to be used for indirect login.

   ```
   GET
   https://LUNAIPADDRESS:PORT/api/lunasa/hsms/{adminHSMid}/partitions/
   {partitionid}/indirect/key?type=legacyKey
   ```

   Output:

   ```
   {
     "exponent": "AQAB",
     "modulus": "tGHiZBb/Ou+VVutU/I9XZhvF410zw307r+..."
   }
   ```

3. Log out from an application partition on adminHSMid.
4. Log in to serviceHSMid as HSM SO (SO).
5. Load the indirect login public key onto the service HSM.

   ```
   POST
   https://LUNAIPADDRESS:PORT/api/lunasa/hsms/{serviceHSMid}/indirect/key
   {
     "exponent":"<as above>",
     "modulus":"<as above>"
   }
   ```

   Output:

   ```
   {
     'Access-Control-Allow-Origin': '*',
     'Content-Type': 'application/json',
   ```

```
  'Location': '/api/lunasa/hsms/{serviceHSMid}/indirect/challenges',
  'Content-Length': '{length}}',
  'Access-Control-Allow-Credentials': 'true'
}
```

6. Log out of serviceHSMid.

Using Indirect Login

1. Log in to application partition of adminHSMid as the crypto officer (CO).
2. Get the token wrapping certificate required for indirect login.

```
GET
https://LUNAIPADDRESS:PORT/api/lunasa/hsms/{adminHSMid}/partitions/
{partitionid}/certificate?type=legacyKey
```

Output:

```
{
  'Access-Control-Allow-Origin': '*',
  'Content-Type': 'application/json',
  'Location': '/api/lunasa/hsms/{adminHSMid}/partitions/{partitionid}/certificate/
{certificateid}',
  'Content-Length': '{length}',
  'Access-Control-Allow-Credentials': 'true'
}
{
  "certificate": "AwAAADCCBAswggHzoAMCAQICAQAwDQYJKoZ..."
}
```

> **NOTE** This object is persistent for the duration of the session. There is no GET partitions/{partitionid}/certificate to obtain a list of objects. The certificate can be retrieved with Luna OpenAPI Reference GET `/api/lunasa/hsms/{adminHSMid}/partitions/{partitionid}/certificate/{certificateid}`.

3. Get the indirect login challenge (certificate) from serviceHSMid.

```
POST
https://LUNAIPADDRESS:PORT/api/lunasa/hsms/{serviceHSMid}/indirect/challenges
{
  "role":"so",
  "ped":"1",
  "certificate":"<as above>"
}
```

Output:

```
{
  'Access-Control-Allow-Origin': '*',
  'Content-Type': 'application/json',
  'Location': '/api/lunasa/hsms/{serviceHSMid}/indirect/challenges/{challengeid}',
  'Content-Length': '{length}',
  'Access-Control-Allow-Credentials': 'true'
}
{
  "challenge": "AAEAAHlUqZ5blhyvdl/bW9EqXwY9xwlVA..."
}
```


> **NOTE** This object is persistent for the duration of the session. There is no GET indirect/challenges to obtain a list of objects. The challenge can be retrieved with GET `/api/lunasa/hsms/{serviceHSMid}/indirect/challenges/{challengeid}`.

4. Get the indirect login response required by serviceHSMid from a user partition on adminHSMid.

```
POST
https://LUNAIPADDRESS:PORT/api/lunasa/hsms/{adminHSMid}/partitions/
{partitionid}/indirect/responses?type=legacyKey
{
  "challenge":"<as above>"
}
```

Output:

```
{
  'Access-Control-Allow-Origin': '*',
  'Content-Type': 'application/json',
  'Location': '/api/lunasa/hsms/{adminHSMid}/indirect/responses/{responseid}',
  'Content-Length': '{length}',
  'Access-Control-Allow-Credentials': 'true'
}
{
  "response": "GZvvxqRYqk6LD3fRKm6MtikoBLjUOsgfMdclectEvoo="
}
```

> **NOTE** This object is persistent for the duration of the session. There is no GET indirect/responses to obtain a list of objects. The response can be retrieved with GET `/api/lunasa/hsms/{serviceHSMid}/indirect/responses/{responseid}`.

5. Use the challenge response to log in to serviceHSMid.

```
POST
https://LUNAIPADDRESS:PORT/api/lunasa/hsms/{serviceHSMid}/login
{
  "response":"<as above>"
}
```

Output:

```
{
  'Access-Control-Allow-Origin': '*',
  'Content-Type': 'application/json',
  'Location': '/api/lunasa/hsms/{adminHSMid}/roles/{roleid}',
  'Content-Length': '{length}',
  'Access-Control-Allow-Credentials': 'true'
}
```

You are now logged into service HSMid as Security Officer ("so").

Indirect Login on FW 7.7 Partitions Using Key

In the examples below, **adminHSMid** refers to the serial number of the HSM (with appliance/firmware 7.7.0 and above) which contains the application partition (v0 or v1) that holds the private key used for indirect login protocol v1.1, and **serviceHSMid** refers to the serial number of the HSM (with appliance/firmware 7.7.0 and above) which contains the administrator partition being configured.

> **NOTE** Indirect Login V1.1 setup cannot be done on Partitions with Partition Version policy value set to 1.

Setting Up Indirect Login

1. Log in to an application partition on adminHSMid as the crypto officer (CO).
2. Export the public key to be used for indirect login v1.1.

```
GET
https://LUNAIPADDRESS:PORT/api/lunasa/hsms/{adminHSMid}/partitions/
{partitionid}/indirect/key?type=key
```

Output:

```
{
  "exponent": "AQAB",
  "modulus": "tGHiZBb/Ou+VVutU/I9XZhvF410zw307r+..."
}
```

3. Log out from an application partition on adminHSMid.
4. Log in to serviceHSMid as HSM SO (SO).
5. Load the indirect login public key onto the service HSM.

```
POST
https://LUNAIPADDRESS:PORT/api/lunasa/hsms/{serviceHSMid}/indirect/key?type=key
{
  "exponent":"<as above>",
  "modulus":"<as above>"
}
```

Output:

```
{
  'Access-Control-Allow-Origin': '*',
  'Content-Type': 'application/json',
  'Location': '/api/lunasa/hsms/{serviceHSMid}/indirect/challenges',
  'Content-Length': '{length}}',
  'Access-Control-Allow-Credentials': 'true'
}
```

6. Log out of service HSMid.

Using Indirect Login

1. Log in to application partition of adminHSMid as the crypto officer (CO).
2. Get the token wrapping certificate required for indirect login.

```
GET
https://LUNAIPADDRESS:PORT/api/lunasa/hsms/{adminHSMid}/partitions/
{partitionid}/certificate?type=key
```

Output:

```
{
  'Access-Control-Allow-Origin': '*',
  'Content-Type': 'application/json',
  'Location': '/api/lunasa/hsms/{adminHSMid}/partitions/{partitionid}/certificate/
{certificateid}',
  'Content-Length': '{length}',
```

```
  'Access-Control-Allow-Credentials': 'true'
}
{
  "certificate": "AwAAADCCBAswggHzoAMCAQICAQAwDQYJKoZ..."
}
```

> **NOTE** This object is persistent for the duration of the session. There is no GET partitions/{partitionid}/certificate to obtain a list of objects. The certificate can be retrieved with GET `/api/lunasa/hsms/{adminHSMid}/partitions/{partitionid}/certificate/{certificateid}`.

3. Get the indirect login challenge (certificate) from serviceHSMid.

```
POST
https://LUNAIPADDRESS:PORT/api/lunasa/hsms/{serviceHSMid}/indirect/challenges
{
  "role":"so",
  "ped":"1",
  "certificate":"<as above>"
}
```

Output:

```
{
  'Access-Control-Allow-Origin': '*',
  'Content-Type': 'application/json',
  'Location': '/api/lunasa/hsms/{serviceHSMid}/indirect/challenges/{challengeid}',
  'Content-Length': '{length}',
  'Access-Control-Allow-Credentials': 'true'
}
{
  "challenge": "AAEAAHlUqZ5blhyvdl/bW9EqXwY9xwlVA..."
}
```

> **NOTE** This object is persistent for the duration of the session. There is no GET indirect/challenges to obtain a list of objects. The challenge can be retrieved with GET `/api/lunasa/hsms/{serviceHSMid}/indirect/challenges/{challengeid}`.

4. Get the indirect login response required by serviceHSMid from a user partition on adminHSMid.

```
POST
https://LUNAIPADDRESS:PORT/api/lunasa/hsms/{adminHSMid}/partitions/
{partitionid}/indirect/responses?type=key
{
  "challenge":"<as above>"
}
```

Output:

```
{
  'Access-Control-Allow-Origin': '*',
  'Content-Type': 'application/json',
  'Location': '/api/lunasa/hsms/{adminHSMid}/indirect/responses/{responseid}',
  'Content-Length': '{length}',
  'Access-Control-Allow-Credentials': 'true'
}
```



```
{
  "response": "GZvvxqRYqk6LD3fRKm6MtikoBLjUOsgfMdclectEvoo="
}
```

> **NOTE** This object is persistent for the duration of the session. There is no GET indirect/responses to obtain a list of objects. The response can be retrieved with GET `/api/lunasa/hsms/{serviceHSMid}/indirect/responses/{responseid}`.

5. Use the challenge response to log in to serviceHSMid.

```
POST
https://LUNAIPADDRESS:PORT/api/lunasa/hsms/{serviceHSMid}/login
{
  "response":"<as above>"
}
```

Output:

```
{
  'Access-Control-Allow-Origin': '*',
  'Content-Type': 'application/json',
  'Location': '/api/lunasa/hsms/{adminHSMid}/roles/{roleid}',
  'Content-Length': '{length}',
  'Access-Control-Allow-Credentials': 'true'
}
```

You are now logged into serviceHSMid as the Security Officer (SO).

Indirect Login on FW 7.7 Partition Using PKC

To make Indirect Login operation eIDAS and CC compliant, the HA Login setup step has changed. Now the PKC (Public Key Confirmation) chain from the HA Login Private Key can be extracted from the primary partition and passed as a parameter to initiate Indirect Login on a secondary partition. This provides some level of assurance that the HA Login private Key resides in an HSM.

In the examples below, **adminHSMid** refers to the serial number of the HSM (with appliance/firmware 7.7.0 and above) which contains the application partition (v0 or v1) that holds the private key used for extracting public key confirmation (pkc) for indirect login protocol v2.0, and serviceHSMid refers to the serial number of the HSM (with appliance/firmware 7.7.0 and above) which contains the administrator partition (v0 or v1) being configured.

Setting Up Indirect Login

1. Log in to an application partition on adminHSMid as the crypto officer (CO).
2. Export the PKC cto be used for indirect login.

```
GET
https://LUNAIPADDRESS:PORT/api/lunasa/hsms/{adminHSMid}/partitions/
{partitionid}/indirect/key?type=pkc
```

Output:

```
{
  "pkcChain": "tGHiZBb/Ou+VVutU/I9XZhvF410zw307r+..."
}
```

3. Log out from an application partition on adminHSMid.
4. Log in to serviceHSMid as HSM SO (SO).

5. Load the indirect login PKC chain onto the service HSM to initialize primary roles for HA Login.

```
POST
https://LUNAIPADDRESS:PORT/api/lunasa/hsms/{serviceHSMid}/indirect/key?type=pkc
{
  "roles":"so,co,cu,lco",
  "pkcChain": "tGHiZBb/Ou+VVutU/I9XZhvF410zw307r+..."
}
```

Output:

```
{
  'Access-Control-Allow-Origin': '*',
  'Content-Type': 'application/json',
  'Location': '/api/lunasa/hsms/{serviceHSMid}/indirect/challenges',
  'Content-Length': '{length}}',
  'Access-Control-Allow-Credentials': 'true'
}
```

6. Log out of serviceHSMid.

Using Indirect Login

1. Log in to application partition of adminHSMid as the crypto officer (CO).
2. Get the token wrapping certificate required for indirect login.

```
GET
https://LUNAIPADDRESS:PORT/api/lunasa/hsms/{adminHSMid}/partitions/
{partitionid}/certificate?type=pkc
```

Output:

```
{
  'Access-Control-Allow-Origin': '*',
  'Content-Type': 'application/json',
  'Location': '/api/lunasa/hsms/{adminHSMid}/partitions/{partitionid}/certificate/
{certificateid}',
  'Content-Length': '{length}',
  'Access-Control-Allow-Credentials': 'true'
}
{
  "certificate": "AwAAADCCBAswggHzoAMCAQICAQAwDQYJKoZ..."
}
```

> **NOTE** This object is persistent for the duration of the session. There is no GET partitions/{partitionid}/certificate to obtain a list of objects. The certificate can be retrieved with GET `/api/lunasa/hsms/{adminHSMid}/partitions/{partitionid}/certificate/{certificateid}`.

3. Get the indirect login challenge (certificate) from serviceHSMid.

```
POST
https://LUNAIPADDRESS:PORT/api/lunasa/hsms/{serviceHSMid}/indirect/challenges
{
  "role":"so",
  "ped":"1",
  "certificate":"<as above>"
}
```

Output:

```
{
  'Access-Control-Allow-Origin': '*',
  'Content-Type': 'application/json',
  'Location': '/api/lunasa/hsms/{serviceHSMid}/indirect/challenges/{challengeid}',
  'Content-Length': '{length}',
  'Access-Control-Allow-Credentials': 'true'
}
{
  "challenge": "AAEAAHlUqZ5blhyvdl/bW9EqXwY9xwlVA..."
}
```

> **NOTE** This object is persistent for the duration of the session. There is no GET indirect/challenges to obtain a list of objects. The challenge can be retrieved with GET `/api/lunasa/hsms/{serviceHSMid}/indirect/challenges/{challengeid}`.

4. Get the indirect login response required by serviceHSMid from a user partition on adminHSMid.

```
POST
https://LUNAIPADDRESS:PORT/api/lunasa/hsms/{adminHSMid}/partitions/
{partitionid}/indirect/responses?type=pkc
{
  "challenge":"<as above>"
}
```

Output:

```
{
  'Access-Control-Allow-Origin': '*',
  'Content-Type': 'application/json',
  'Location': '/api/lunasa/hsms/{adminHSMid}/indirect/responses/{responseid}',
  'Content-Length': '{length}',
  'Access-Control-Allow-Credentials': 'true'
}
{
  "response": "GZvvxqRYqk6LD3fRKm6MtikoBLjUOsgfMdclectEvoo="
}
```

> **NOTE** This object is persistent for the duration of the session. There is no GET indirect/responses to obtain a list of objects. The response can be retrieved with GET `/api/lunasa/hsms/{serviceHSMid}/indirect/responses/{responseid}`.

5. Use the challenge response to log in to serviceHSMid.

```
POST
https://LUNAIPADDRESS:PORT/api/lunasa/hsms/{serviceHSMid}/login
{
  "response":"<as above>"
}
```

Output:

```
{
  'Access-Control-Allow-Origin': '*',
  'Content-Type': 'application/json',
  'Location': '/api/lunasa/hsms/{adminHSMid}/roles/{roleid}',
  'Content-Length': '{length}',
  'Access-Control-Allow-Credentials': 'true'
}
```

You are now logged into serviceHSMid as the Security Officer (SO).

CHAPTER 5: Status Codes

This page summarizes the status codes that the REST API can return. Refer to specific resources for details on code status and interpretation.

Code	Status
200	Success
201	Success; a new resource was created
202	Task generated
204	Change successful; no content returned
303	Task finished successfully or with error
400	Request failed due to malformed request. Possible reasons: resource, parameters and/or headers
401	Unauthorized
404	Request failed due to resource not being found
500	REST API framework failed to complete action for resource
501	REST API framework does not support the requested action on the resource

CHAPTER 6: Block List

The webserver includes a block list which stops malicious users from making requests to the REST API. The system analyzes a user's request and detects malicious patterns. Once blocklisted, a user's IP will be blocked at the TCP socket level and will not be allowed a connection. All block list activity is logged in the lunalogs log file.

The following table lists blocklistable offences and the number of infractions caused by each:

Offence	Number of Infractions
Bad login credential	2
Request timeout	5
Certificate renegotiation	1
Bad request payload	1

Advanced Block List Usage

The block list can be configured by specifying a severity percentage or by modifying the attributes directly.

Severity

The simplest way to configure the block list is to assign a severity percentage between 0 and 100, where 0 turns the block list off and 100 is the highest possible severity. The default value is 50; this value is recommended as being generally lenient while effective at stopping malicious users.

Attributes

The block list is controlled by three attributes: **maxInfractionCount**, **timeoutStart** and **timeoutMultiplier**.

- **maxInfractionCount**: the number of infractions committed before a user is blocklisted. It is represented as an internal counter per user. When an infraction is committed, the counter is incremented. The user is blocklisted when the maximum number is reached.
- **timeoutStart**: the time (in seconds) before the infraction count is updated. Each user has an update timer which begins at this value and increments with each infraction (see timeout multiplier). When the time expires, the infraction counter for the current user is decreased. When the infraction counter returns to zero, the timer is reset to this value.
- **timeoutMultiplier**: the amount the timeout multiplies with each infraction.

Example: The webserver's block list is configured as follows:

- **maxInfractionCount** = 5
- **timeoutStart** = 1
- **timeoutMultiplier** = 2

A user attempts to log in, but the credentials are incorrect. Login with bad credentials counts as 2 infractions, so the infraction count is reduced by 2, bringing it down from 5 to 3.

The timeout counter is incremented by the multiplier, making the timeout 4 seconds (timeout x multiplier ^ infractions = 1 x 2 ^ 2). The user must now wait 4 seconds for their infraction count to increase to 4 from 3. The timeout remains at 4 seconds until the next infraction is committed.

If the user waits an additional 4 seconds, the infraction count will be back to its default, and the timeout is reset to its **timeoutStart** value.

A malicious user may send requests at a more aggressive rate. If the user attempts to make 10 bad requests, 5 come through, and the user is immediately blocklisted.

The timeout counter is now set to 32 seconds (timeout x multiplier ^ infractions = 1 x 2 ^ 5). The user waits 32 seconds, and the infraction count is set to 1.

The user then performs another bad request and is blocklisted for 64 seconds (timeout x multiplier ^ infractions = 32 x 2 ^ 1).

CHAPTER 7: Critical Operation

This section provides information about critical operations that are identified as essential and cannot be executed simultaneously on the Luna Network HSM appliance using REST API:

These operations are listed below:

- Appliance Power Off (see POST /api/lunasa/actions/powerOff)
- Appliance Reboot (see POST /api/lunasa/actions/reboot)
- Backup system configuration (see POST /api/lunasa/config/backups)
- Factory Reset system configuration (see POST /api/lunasa/config/factoryReset)
- Firmware Upgrade (see POST /api/lunasa/hsms/{hsmid}/firmware/actions/upgrade)
- Firmware Rollback (see POST /api/lunasa/hsms/{hsmid}/firmware/actions/rollback)
- Package update and verify (see POST /api/lunasa/packageFiles)
- Restore system configuration (see POST /api/lunasa/config/backups/{backupid}/actions/restore)

CHAPTER 8: Protecting Resources

Some REST API actions can lead to destructive outcomes. For example:

- HSM zeroization: all partitions and cryptographic material is destroyed
- Applying destructive policy settings: all cryptographic material is destroyed
- Exceeding last login attempt: all cryptographic material is destroyed
- Applying destructive HSM capability update: all cryptographic material is destroyed

To provide an application with a measure of control over these kind of actions, the REST API has a process to protect destructive resources. This two-step process is as follows:

1. The application requests the destructive resource. This action creates a suspended (WAITING) task.
2. The application kick-starts the task so that it can run and complete the destructive process.

An application performs the kick-start via a task actions resource called **start** as shown by the following prototype:

```
POST
https://LUNAIPADDRESS:PORT/tasks/{taskid}/actions/start
```

Additionally, the REST API has another resource to list all task actions:

```
GET
https://LUNAIPADDRESS:PORT/tasks/{taskid}/actions
```

Example: The following example describes the steps required to zeroize the HSM:

1. Log in to REST.
2. Log in to HSM as SO.
3. POST /api/lunasa/hsms/{hsmid}/actions/zeroize

   Returned is a task object referenced by **/tasks/{taskid}**
4. GET /tasks/{taskid} and confirm the task is in WAITING (suspended) state.
5. POST /tasks/{taskid}/actions/{actionid} to continue the zeroization.
6. GET /tasks/{taskid} to confirm the task is in FINISHED or ERROR state.
7. GET /tasks/{taskid}/response to obtain the output of the zeroization resource.

CHAPTER 9: Formatting

The REST API framework provides formatting options for requests and their responses to suit the needs of your application.

Responses

To format a response, include your formatting options in the query, as described below.

Limit

This is a filter that limits the number of elements in an array. If, for example, the server would normally return 100 elements (0-99), setting a limit of 20 would return elements 0-19 only: `/verbatim https://LUNAIPADDRESS:PORT/api/hsms/{hsmid}/partitions?limit=20 /endverbatim`. When limit is used with offset, it can generate a link header with a link to the next page. The link header will not be created if there are no more pages.

Offset

This filter offsets the results of an array. If, for example, the server would normally return 100 elements (0-99), an offset of 20 would return elements 20-99: `/verbatim api/hsms/1234/partitions?offset=20 /endverbatim`.

> **NOTE**
>
> - Any filter that operates on arrays will modify the first array in an object. For example, if you expand partitions when fetching an HSM, the partitions are formatted.
> - Multiple filters can be combined for desired effect. For example:
>
> `api/hsms/1234/partitions?limit=2&offset=2`
>
> This format will return elements 2 and 3 but skip 0 and 1.

Requests

The REST API framework provides some options for formatting requests.

URL Encoding

URLs can only be sent using the ASCII character-set. Since the REST API framework allows you to use some non-ASCII characters in URLs, these characters must be encoded by the REST client and decoded by the server. Currently, the REST API framework supports the ASCII characters from 20 (space) to 126 (tilde).

```
GET
https://LUNAIPADDRESS:PORTapi/lunasa/hsms/1234/partitions/20160901/stc/clients/client%20%luna
```

The example above encodes the client name **client luna** to **client%20luna** in URL.

CHAPTER 10: Headers

Headers are used by the server to process requests.

Content-Type

Content-Type defines the type of content the server should expect for a request so that it can be processed appropriately. You must specify this type when using PUT, PATCH, and POST requests.

The template for the **Content-Type** header is defined as:

```
{
    "Content-Type" : "application/vnd.safenetinc.lunasa+{type};version={version}"
}
```

Types

Currently the REST API supports the following types:

- **json**: sending json data to the server.
- **octet-stream**: sending a stream of data to the server.
- **multipart**: sending multipart data to the server.

Version

Version is a number defining the version of the resource to access. It is good practice to specify the current REST API version you wish to use; the behavior of your applications will remain consistent even if the resource changes in a later version. If you do not specify a version, REST API defaults to the newest version.

Accept-Type

The **Accept-Type** header entry is defined the same way as the **Content-Type**; it should be specified when using GET and DELETE requests.

> **NOTE** If both the **Content-Type** and **Accept-Type** headers are given, the **Content-Type** will be used.

> **NOTE**
>
> - It is not a mandate for all endpoints to support these values uniformly. Instead, the compatibility of these values depends on the nature of the input data.
> - Should any endpoint documentation explicitly specify the use of **octet-stream** or **multipart**, then it is advisable to use these values exclusively in the **Content-Type** header.
> - If **octet-stream** or **multipart** is specified in the content-type, it is imperative that the input data be transmitted accordingly in the request body. Failure to adhere to this requirement will result in a FRAMEWORK_BAD_REQUEST response.

CHAPTER 11: File I/O

The REST API supports file input and output. This allows you to send and receive files within requests and responses.

Receiving Files

When you receive a file, the response object will contain the contents of the file in a buffer that can then be iterated and saved to a file.

Example:

```
 r = requests.get("/api/lunasa/webServer/config/csr",
                     stream=True,
                     cookies=cookies,
                     verify=False,
                     allow_redirects=False)

    with open("ssl.csr", 'wb') as csr:
        for chunk in r.iter_content(chunk_size=1024):
            if chunk:
                csr.write(chunk)
```

An alternative way to receive a file is to provide the request **Accept** header with the value **application/octet-stream**. This is not always required; the method above accounts for most cases. Some resources, however, may return json as well as octet-stream, in which case the **Accept** header is required.

Example:

```
headers["Accept"] = "application/octet-stream"
  r = requests.get("/api/lunasa/partitionPolicyTemplates/myTemplate",
                     cookies=cookies,
                     verify=False,
                     headers=headers,
                     allow_redirects=False)

    with open("template.csv", 'wb') as csr:
      for chunk in r.iter_content(chunk_size=1024):
          if chunk:
              csr.write(chunk)
```

Sending Files

To send a file, the **Content-Type** header needs to be set to **octet-stream** to notify the server that it will be receiving a file. In python, passing the file object to the data parameter is all that is required.

Header format:

```
headers = {'Content-Type': "application/vnd.safenetinc.lunasa+octet-stream;version="}
```

Example:

```
with open(filename, 'rb') as payload:
        r = requests.put("/api/lunasa/webServer/config/certificate",
                         stream=True,
```

```
cookies=cookies,
data=payload,
header=headers,
verify=False,
allow_redirects=False)
```

CHAPTER 12: SNMP Configuration Guide

This document provides step-by-step instructions for enabling, configuring, and verifying SNMP (Simple Network Management Protocol) functionality between a Luna Network HSM and an SNMP Manager running on Linux. It is broken down into the following sections:

Introduction to Traps and Informs

In SNMP, traps and informs are notifications sent from the Luna Network HSM (agent) to the SNMP Manager to report events or status changes The key difference is that traps are one-way messages sent without acknowledgment while informs require the SNMP manager to acknowledge receipt; if no acknowledgment is received, the HSM resends the message. This makes informs more reliable but slightly slower due to the confirmation process. In general, traps are suitable for routine monitoring while informs are preferred for critical alerts where confirmation of delivery is important.

Overview of SNMP Functionality

The Luna Network HSM supports SNMPv3, providing secure communication and monitoring capabilities for HSM status, operational metrics, and event traps.

The configuration process includes:

> Creating SNMP users on the HSM
> Defining traps (destinations for SNMP notifications)
> Setting up SNMP notifications
> Configuring the SNMP Manager to receive and interpret these traps

Prerequisites

On SNMP Manager (Linux):

> Root or sudo privileges
> Network connectivity with the HSM (UDP ports 161 and 162 open)

Configuring SNMP on the SNMP Manager (Linux Host)

1. Install required packages,

```
yum install net-snmp-devel net-snmp net-snmp-utils -y
```

> **net-snmp**: Core SNMP daemon and libraries.

> **net-snmp-devel**: Provides net-snmp-config utility.

> **net-snmp-utils**: Provides SNMP testing tools like snmpwalk, snmptable, etc.

2. Locate and install MIB files.

```
net-snmp-config --default-mibdirs
# Expected output (example): /usr/share/snmp/mibs

cp <Luna MIB's path> /usr/share/snmp/mibs

snmptranslate -Tp
# Displays MIB hierarchy including CHRYSALIS-UTSP-MIB, SAFENET-HSM-MIB, and SAFENET-
APPLIANCE-MIB
```

3. Congifure the SNMP trap daemon.

   The trap daemon (snmptrapd) listens for SNMP traps on UDP port 162.

   Edit the configuration file /etc/snmp/snmptrapd.conf and add the following lines:

```
# SNMPv3 user configuration for receiving traps
createUser restuser SHA myAuthPassword AES myPrivPassword
# createUser <securityName> <authenticationProtocol> <authenticationPassword>
<privacyProtocol> <privacyPassword>

authUser log,execute,net restuser
# Allows this SNMPv3 user to log traps

outputOption s
# Logs will go into /var/log/messages
```

4. Start and enable snmptrapd service

```
sudo systemctl enable snmptrapd
sudo systemctl start snmptrapd
sudo systemctl status snmptrapd

# Ensure it listens on UDP port 162
sudo netstat -anu | grep 162
```

Configuring SNMP on the Luna Network HSM Device

1. Start the SNMP service.

```
service start snmp
```

2. Create SNMP user.

   Use the REST API endpoint to create an SNMPv3 user.

```
POST /api/lunasa/snmp/users

{
  "securityName": "restuser",
  "authenticationPassword": "password",
```

```
  "authenticationProtocol": "SHA",
  "privacyPassword": "password",
  "privacyProtocol": "AES"
}
```

3. Create SNMP trap destination.

   Define where the HSM should send SNMP traps (to SNMP Manager).

   ```
   POST /api/lunasa/snmp/traps

   {
     "address": "<SNMP Manager IP>",
     "securityName": "restuser",
     "authenticationPassword": "password",
     "authenticationProtocol": "SHA",
     "privacyPassword": "password",
     "privacyProtocol": "AES",
     "engineId": "<engineId from SNMP Manager>",
     "trapType": "trap"
   }
   ```

   Verify the created trap:

   ```
   GET /api/lunasa/snmp/traps
   ```

4. Add notification for SNMP user.

   Each user can have one or more notifications defined.

   Example for trap notification:

   ```
   POST /api/lunasa/snmp/users/{userId}/notifications

   {
     "address": "<SNMP Manager IP>",
     "port": 162,
     "authenticationPassword": "password",
     "authenticationProtocol": "SHA",
     "privacyPassword": "password",
     "privacyProtocol": "AES",
     "type": "trap",
     "engineId": "000FFFFFFFA9"
   }
   ```

   Example for inform notification:

   ```
   {
     "address": "<SNMP Manager IP>",
     "port": 162,
     "authenticationPassword": "password",
     "authenticationProtocol": "SHA",
     "privacyPassword": "password",
     "privacyProtocol": "AES",
     "type": "inform"
   }
   ```

Testing SNMP Functionality

Using MIB files:

```
# CHRYSALIS-UTSP-MIB Examples
snmpwalk -v3 -l authPriv -u <user> -a SHA -A <authPass> -x AES -X <privPass> {HSM_IP}
CHRYSALIS-UTSP-MIB::hsmOperationRequests
snmpwalk -v3 -l authPriv -u <user> -a SHA -A <authPass> -x AES -X <privPass> {HSM_IP}
CHRYSALIS-UTSP-MIB::hsmCriticalEvents

# SAFENET-HSM-MIB Examples
snmpwalk -v3 -l authPriv -u <user> -a SHA -A <authPass> -x AES -X <privPass> {HSM_IP}
SAFENET-HSM-MIB::hsmTable
snmpwalk -v3 -l authPriv -u <user> -a SHA -A <authPass> -x AES -X <privPass> {HSM_IP}
SAFENET-HSM-MIB::hsmPartitionTable

# SAFENET-APPLIANCE-MIB Example
snmpwalk -v3 -l authPriv -u <user> -a SHA -A <authPass> -x AES -X <privPass> {HSM_IP}
SAFENET-APPLIANCE-MIB::appSoftwareVersion
```

Testing Traps

After configuring both the Luna Network HSM (SNMP Agent) and the SNMP Manager (trap receiver), perform the following steps to test SNMP traps.

1. Enable debug logging on SNMP Manager.

   To confirm that traps are being recieved, start `snmptrapd` in the foreground with verbose logging:

   ```
   sudo systemctl stop snmptrapd
   sudo snmptrapd -f -Lo -d -n
   ```

2. Generate a test trap from the Luna Network HSM 7

   Before generating a test SNMP trap, regenerate the NTLS certificate with an expiry period of within 5 days, and then disable and then enable the NTLS certificate monitor using the following commands:

   ```
   ntls certificate monitor disable
   ntls certificate monitor enable
   ```

3. Verify trap reception on SNMP Manager.

   ```
   sysUpTimeInstance = Timeticks: (44818015) 5 days, 4:29:40.15
   snmpTrapOID.0 = OID: enterprises.12383.3.1.3.1
   enterprises.12383.3.1.2.6 = STRING: "Nov  6 15:28:00 2025 GMT"
   ```

See Also

SNP Trap Configuration: POST /api/lunasa/snmp/traps

SNP User Configuration: POST /api/lunasa/snmp/users

SNP User Notification Creation: POST /api/lunasa/snmp/users/{userid}/notifications

CHAPTER 13: Syslog Encryption

TLS support is added to the Luna Network HSM 7 syslog implementation to encrypt log messages being sent to a remote server. This improves security of your logs by preventing their interception during transit. Such protection is desirable to safeguard details that could reveal the current state of the appliance.

The primary focus of the feature is to support the following use cases.

- Server authentication with self-signed certificates
- Server authentication with CA signed certificates
- Mutual authentication with self-signed certificates
- Mutual authentication with CA signed certificates

> **NOTE** When configuring two or more remote servers, certificates from multiple chains of trust are not supported. All server and client certificates must be signed by the same entity. This limitation is tied to the existing version of rsyslog supported by CentOS.

Server Authentication with Self-Signed Certificates

1. Server generates a private key and self-signed certificate.
2. Server passes the root certificate to the LNH.
3. LNH adds this certificate to its trust store.

```
POST
https://LUNAIPADDRESS:PORT/api/lunasa/syslog/remoteHosts/ca
{
    "file": "sample_ca.pem"
}
```

4. User configures server information.

```
POST
https://LUNAIPADDRESS:PORT/api/lunasa/syslog/remoteHosts
{
    "address": "1.2.3.4",
    "protocol": "tcp",
    "port": 514,
    "mode": "server",
    "name": "1.2.5.6",
    "tls": true
}
```

Server Authentication with CA Signed Certificates

1. Server generates a private key and CSR and gets the CSR signed from the CA.
2. Server adds the acquired certificate.
3. LNH adds the CA certificate to its trust store.

```
POST
https://LUNAIPADDRESS:PORT/api/lunasa/syslog/remoteHosts/ca
{
    "file": "sample_ca.pem"
}
```

4. User configures server information.

```
POST
https://LUNAIPADDRESS:PORT/api/lunasa/syslog/remoteHosts
{
    "address": "1.2.3.4",
    "protocol": "tcp",
    "port": 514,
    "mode": "server",
    "name": "1.2.5.6", M
    "tls": true
}
```

Mutual Authentication with Self-Signed Certificates

1. Server and the LNH each generate a private key and self-signed certificate.

```
At LNH:
PUT | PATCH
https://LUNAIPADDRESS:PORT/api/lunasa/syslog/remoteHosts/certificate
{
    "cn": "1.2.3.4",
    "startDate": "2024-02-12",
    "days": 300,
    "country": "CA",
    "state": "Ontario",
    "location": "Ottawa",
    "organization": "Thales",
    "orgunit": "GPHSM",
    "email": "sample@email.com",
    "subjectAltNames": ["IP:1.2.3.4", "DNS:example.com"],
    "keySize": 2048,
    "keyType": "rsa"
}
```

2. Server sends the self-signed certificate to the LNH, and the LNH adds the acquired certificate.

```
POST
https://LUNAIPADDRESS:PORT/api/lunasa/syslog/remoteHosts/ca
{
    "file": "sample_ca.pem"
}
```

3. The LNH sends the self-signed certificate to the server, and the server adds the acquired certificate.

```
[CAserver]# scp client_sign.pem operator@192.168.14.93:
```

4. User configures server information.

```
POST
https://LUNAIPADDRESS:PORT/api/lunasa/syslog/remoteHosts
{
    "address": "1.2.3.4",
    "protocol": "tcp",
    "port": 514,
    "mode": "server",
```

```
    "name": "1.2.5.6",
    "tls": true
}
```

Mutual Authentication with CA Signed Certificates

1. Server and the LNH each generate a private key and CSR.

```
At LNH:
POST
https://LUNAIPADDRESS:PORT/api/lunasa/syslog/remoteHosts/csr
{
    "cn": "1.2.3.4",
    "startDate": "2024-02-12",
    "days": 300,
    "country": "CA",
    "state": "Ontario",
    "location": "Ottawa",
    "organization": "Thales",
    "orgunit": "GPHSM",
    "email": "sample@email.com",
    "subjectAltNames": ["IP:1.2.3.4", "DNS:example.com"],
    "keySize": 2048,
    "keyType": "rsa"
}
```

2. Server and the LNH add the received signed certificates.

```
PUT
https://LUNAIPADDRESS:PORT/api/lunasa/syslog/remoteHosts/certificate
{
    "file": "sample_certificate.pem"
}
```

3. Server and LNH add the CA certificate to their trust store.

```
POST
https://LUNAIPADDRESS:PORT/api/lunasa/syslog/remoteHosts/ca
{
    "file": "sample_ca.pem"
}
```

4. User configures server information.

```
POST
https://LUNAIPADDRESS:PORT/api/lunasa/syslog/remoteHosts
{
    "address": "1.2.3.4",
    "protocol": "tcp",
    "port": 514,
    "mode": "server",
    "name": "1.2.5.6",
    "tls": true
}
```

CHAPTER 14: Tasks

Many administrative actions on the appliance take time to complete. Updating the HSM firmware, for example, can take several minutes. Instead of simply blocking further actions during this time, the REST API creates tasks for time-consuming resources and returns an immediate response for the action. An application can monitor the state of an associated task and perform different actions accordingly. While the firmware is updating, for example, an application might display one of the following:

- An hourglass, signifying that the operation is still in progress
- A checkmark for a successful update
- An X for a failed update

The state obtained for a GET operation on the applicable task identifier provides the information to decide what follow-up action to take.

Also See: Task Management Flowchart.

How to Use Tasks

This example describes one way that an application can use a task (logging in to the HSM) and the state returned by the server on query.

Post the login resource on the HSM:

```
POST
https://LUNAIPADDRESS:PORT/api/lunasa/hsms/151256/login
{
  "password": "Test@123",
  "role": "so"
}
```

You get back:

```
{
    "finishTime": "",
    "instance": "/tasks/3",
    "responseUrl": "/tasks/3/response",
    "sourceUrl": "/api/lunasa/hsms/151256/login",
    "startTime": "",
    "state": "Waiting",
    "details": ""
}
```

To obtain a list of tasks from the appliance, get the tasks:

```
GET
https://LUNAIPADDRESS:PORT/tasks
```

The server response for our example might include:

```
{
    "tasks": [
        {
            "finishTime": "",
            "instance": "/tasks/6",
            "responseUrl": "/tasks/6/response",
```



```
            "sourceUrl": "",
            "startTime": "2015-07-05T06:53:36Z",
            "state": "Running",
            "details":""
        }
    ]
}
```

Running means that the HSM login action is still in progress. After login is completed, a query of tasks returns:

```
{
    "tasks": [
        {
            "finishTime": "2015-07-05T06:53:49Z",
            "instance": "/tasks/6",
            "responseUrl": "/tasks/6/response",
            "sourceUrl": "/api/lunasa/hsms/151256/login",
            "startTime": "2015-07-05T06:53:36Z",
            "state": "Finished",
            "details":""
        }
    ]
}
```

Starting a new login operation and using the task instance returned in the server response for a GET operation shows:

```
GET
https://LUNAIPADDRESS:PORT/tasks/7
...
{
    "finishTime": "",
    "instance": "/tasks/7",
    "responseUrl": "/tasks/7/response",
    "sourceUrl": "/api/lunasa/hsms/151256/login",
    "startTime": "2015-07-05T09:10:30Z",
    "state": "Running",
    "details":""
}
```

Periodically polling with a GET on this resource returns a **Running** state. If the action fails (due to no PED response, for example), the server response is:

```
GET
https://LUNAIPADDRESS:PORT/tasks/7
...
    "finishTime": "2015-07-05T09:15:30Z",
    "instance": "/tasks/7",
    "responseUrl": "/tasks/7/response",
    "sourceUrl": "/api/lunasa/hsms/151256/login",
    "startTime": "2015-07-05T09:10:30Z",
    "state": "Error",
    "details":""
}
```

You might encounter the following other states:

- **Waiting** - The REST API is blocked from handing off a request to a plugin to complete. For example, with multi-party authentication, login to the REST API reports this state until authentication is established.

> **NOTE** Under high load conditions, when the server is occupied with ongoing tasks, a request intended to retrieve a **Waiting** task may temporarily return a **Running** task. Once the server has sufficient resources to handle the request, it will correctly return a new **Waiting** task, which can then be started/handled by the user through the standard workflow.

- **Cancelled** - This state is reserved for future use and is TBD until then.
- **Timed Out** - This state is reserved for future use and is TBD until then.
- **Skipped** - This state is reserved for future use and is TBD until then.

> **NOTE** Applications may choose to clean up stale tasks (tasks in **Waiting**, **Finished**, and **Error** states that must be started or queried in order to be removed) using the delete task resources (DELETE /tasks/{taskid} and DELETE /tasks). Cleanup is not required, however; a maximum of 20 stale tasks is allowed.

Commonly Tasked Resources

Any resource action can result in a task. The return code for a tasked action on a resource is 202. Some resource actions are more likely to always use tasks to track progress, specifically:

- PUT /api/lunasa/hsms/{hsmid}
- POST /api/lunasa/hsms/{hsmid}/partitions/{partitionid}/object/actions/backup
- POST /api/lunasa/hsms/{hsmid}/partitions/{partitionid}/object/actions/restore
- DELETE /api/lunasa/hsms/{hsmid}/partitions/{partitionid}/object/objects
- POST /api/lunasa/hsms/{hsmid}/partitions
- PUT /api/lunasa/hsms/{hsmid}/partitions/{partitionid}
- PUT /api/lunasa/hsms/{hsmid}/partitions/{partitionid}/roles/{roleid}
- PUT /api/lunasa/hsms/{hsmid}/partitions/{partitionid}/policies/{policyid}
- PATCH /api/lunasa/hsms/{hsmid}/partitions/{partitionid}/policies/{policyid}
- POST /api/lunasa/hsms/{hsmid}/ped/peds/{pedid}/actions/connect
- POST /api/lunasa/hsms/{hsmid}/ped/peds/{pedid}/actions/disconnect
- POST /api/lunasa/hsms/{hsmid}/firmware/actions/rollback
- POST /api/lunasa/hsms/{hsmid}/firmware/actions/upgrade
- PUT /api/lunasa/hsms/{hsmid}/policies/{policyid}
- PATCH /api/lunasa/hsms/{hsmid}/policies/{policyid}

Any resource that uses the PIN entry device is tasked. For example,POST /api/lunasa/hsms/{hsmid}/login . Some reference pages show examples of how tasks might result for certain operations.

