gRPC Authentication

gRPC authentication modes

A gRPC authentication mode is a security mechanism for gRPC communication that

- provides different methods to verify the identity of clients and servers,
- supports both metadata-based and certificate-based approaches for authentication, and
- enables compliance with varying security requirements through configurable settings such as TLS, Mutual TLS, and non-TLS options.

This section details the authentication modes supported by gRPC to secure communication and ensure authorized access to services.

gRPC supports multiple authentication modes to secure communication between clients and servers. These modes ensure that only authorized entities can access gRPC services such as gNOI, gRIBI, and P4RT. Upon receiving a gRPC request, the device authenticates the user and performs authorization checks.

The following table lists the authentication types and their configuration requirements:

Table 1: gRPC authentication modes and configuration requirements

Type	Authentication Method	Authorization Method	Configuration Requirement	Requirement From Client
Metadata with TLS	username, password	username	**grpc**	username, password, and CA
Metadata without TLS	username, password	username	**grpc no-tls**	username, password
Metadata with Mutual TLS	username, password	username	**grpc tls-mutual**	username, password, client certificate, client key, and CA

Type	Authentication Method	Authorization Method	Configuration Requirement	Requirement From Client
Certificate based Authentication	client certificate's common name field	username from client certificate's common name field	**grpc tls-mutual** and **grpc certificate authentication**	client certificate, client key, and CA

Certificate-based authentication

In Extensible Manageability Services (EMS) gRPC, certificates play a vital role in ensuring secure and authenticated communication. The EMS gRPC utilizes these certificates for authentication:

```
/misc/config/grpc/ems.pem
                    /misc/config/grpc/ems.key
                    /misc/config/grpc/ca.cert
```

Note For clients to use the certificates, ensure to copy the certificates from **/misc/config/grpc/**

Generation of certificates

These certificates are typically generated using a Certificate Authority (CA) by the device. The EMS certificates, including the server certificate (**ems.pem**), public key (**ems.key**), and CA certificate (**ca.cert**), are generated with specific parameters like the common name **ems.cisco.com** to uniquely identify the EMS server and placed in the **/misc/config/grpc/** location.

The default certificates that are generated by the server are Server-only TLS certificates and by using these certificates you can authenticate the identity of the server.

Usage of certificates

These certificates are used for enabling secure communication through Transport Layer Security (TLS) between gRPC clients and the EMS server. The client should use **ems.pem** and **ca.cert** to initiate the TLS authentication.

To update the certificates, ensure to copy the new certificates that have been generated earlier to the location and restart the server.

Custom certificates

If you want to use your own certificates for EMS gRPC communication, then you can follow a workflow to generate custom certificates with the required parameters and then configure the EMS server to use these custom certificates. This process involves replacing the default EMS certificates with the custom ones and ensuring that the gRPC clients also trust the custom CA certificate. For more information on how to customize the **common-name**, see *Certificate Common-Name For Dial-in Using gRPC Protocol*.

Configure authentication for gRPC services

This task explains how to configure different types of authentication for gRPC services, including TLS and AAA-based authentication.

Before you begin

Ensure that the router supports gRPC and that you have access to the CLI in configuration mode. TLS and AAA configurations must be available if required by the authentication method.

Starting from release 24.4.1, the gRPC server supports TLS version 1.3

Procedure

Step 1 Configure your preferred authentication method:

- Configure authentication using metadata with TLS

```
Router#config
Router(config)#grpc
Router(config-grpc)#commit
```

- Configure authentication using metadata without TLS

```
Router#config
Router(config)#grpc
Router(config-grpc)#no-tls
Router(config-grpc)#commit
```

- Configure authentication using metadata with mutual TLS

```
Router#config
Router(config)#grpc
Router(config-grpc)#tls-mutual
Router(config-grpc)#commit
```

- Configure certificate-based authentication

```
Router(config)#grpc
Router(config-grpc)#tls-mutual
Router(config-grpc)#certificate-authentication
Router(config-grpc)#commit
```

Step 2 Verify the configuration.

Example:

```
Router# show grpc
Tue Jul 30 09:54:23.001 UTC

Server name                             : DEFAULT
Address family                          : dual
Port                                    : 57400

Service ports
  gNMI                                  : none
  P4RT                                  : none
  gRIBI                                 : none

DSCP                                    : Default
TTL                                     : 64
VRF                                     : global-vrf
Server                                  : enabled
TLS                                     : enabled
TLS mutual                              : disabled
Trustpoint                              : none
Certificate Authentication              : disabled
```

```
Certificate common name                 : ems.cisco.com
TLS v1.0                                : enabled
Maximum requests                        : 128
Maximum requests per user               : 10
Maximum streams                         : 32
Maximum streams per user                : 32
Maximum concurrent streams              : 32
Memory limit (MB)                       : 1024
Keepalive time                          : 30
Keepalive timeout                       : 20
Keepalive enforcement minimum time      : 300



TLS cipher suites
  Default                               : none
  Default TLS1.3                        : aes_128_gcm_sha256
                                        : aes_256_gcm_sha384
                                        : chacha20_poly1305_sha256

  Enable                                : none
  Disable                               : none

  Operational enable                    : ecdhe-rsa-chacha20-poly1305
                                        : ecdhe-ecdsa-chacha20-poly1305
                                        : ecdhe-rsa-aes128-gcm-sha256
                                        : ecdhe-ecdsa-aes128-gcm-sha256
                                        : ecdhe-rsa-aes256-gcm-sha384
                                        : ecdhe-ecdsa-aes256-gcm-sha384
                                        : ecdhe-rsa-aes128-sha
                                        : ecdhe-ecdsa-aes128-sha
                                        : ecdhe-rsa-aes256-sha
                                        : ecdhe-ecdsa-aes256-sha
                                        : aes128-gcm-sha256
                                        : aes256-gcm-sha384
                                        : aes128-sha
                                        : aes256-sha
  Operational disable                   : none
Listen addresses                        : ANY
```

The gRPC service is configured with the selected authentication method and is ready to accept secure client connections.

What to do next

Verify the gRPC connection and monitor authentication logs to ensure proper access control.

gRPC servers with TLS version 1.3 support

gRPC servers with TLS version 1.3 support are network security solutions that

- provide end-to-end encrypted communication between clients and servers,
- use modern cryptographic protocols for stronger security and performance, and
- allow administrators to configure minimum and maximum TLS versions for compliance and interoperability.

Table 2: Feature History Table

Feature Name	Release Information	Description
gRPC Server TLS Version 1.3 Support	Release 24.4.1	You can now enhance the security of your network connections with stronger protection against vulnerabilities by enabling TLS 1.3 support over gRPC services. This update improves performance with faster connection times and reduced latency by reducing the number of round trips required to establish a connection and removing outdated ciphers. Additionally, it complies with internal security mandates, providing a more robust and future-proof solution for your network management needs. Previously, gRPC server supported TLS version 1.2. The feature introduces these changes: **CLI:** • **tls-min-version** • **tls-max-version**

Security benefits of TLS 1.3

The gRPC Remote Procedure Calls (gRPC) server Transport Layer Security (TLS) version 1.3 support is a security feature that:

- Provides end-to-end communications security over networks
- Prevents unauthorized access and eavesdropping
- Protects against tampering and message forgery

The TLS private key is encrypted before being stored on the disk. For more details on SSL or TLS version certificates, keys, and communication parameters, see Manage certificates using Certz.proto.

Guidelines and limitations for TLS configuration

TLS version configuration limitations

- Ensure that the `tls-min-version` value is not greater than the `tls-max-version` value.
- Starting in Release 2.4.4.1, the `tlsv1-disable` command is deprecated. Avoid using this command in new configurations.
- If you use the `tlsv1-disable` command, do not use the `tls-min-version` or `tls-max-version` commands.
- If you use the `tls-min-version` and `tls-max-version` commands, do not use the `tlsv1-disable` command.

Best practice for disabling TLS 1.0

To disable TLS version 1.0, set the `tlsv1-disable` command. Alternatively, you can set the `tls-min-version` to a value greater than 1.0.

Configure gRPC TLS version

Configuring gRPC TLS version enables you to control which TLS protocol versions are permitted for secure gRPC communication between the router and clients. This can be important for maintaining compatibility and achieving desired security standards.

Before you begin

- Verify that gRPC is enabled on the router.
- Determine which TLS versions (1.0, 1.1, 1.2, or 1.3) your environment and clients require.

Procedure

Step 1 Configure gRPC TLS minimum, maximum, or both versions.

Example:

- Configure gRPC TLS maximum version.

```
Router# config
                                Router(config)# grpc
                                Router(config-grpc)# tls-max-version 1.2
                                Router(config-grpc)# commit
```

tls-max-version can be 1.0, 1.1, 1.2, or 1.3. The default maximum version for TLS is 1.3.

Step 2 Verify the gRPC TLS minimum and maximum versions.

Example:

```
Router# show grpc
                        Thu Aug 29 00:49:24.428 UTC

                        Server name                              : DEFAULT
                        Address family                           : dual
                        Port                                     : 57400

                        Service ports
                        gNMI                                   : none
                        P4RT                                   : none
                        gRIBI                                  : none

                        DSCP                                     : Default
                        TTL                                      : 64
                        VRF                                      : global-vrf
                        Server                                   : disabled (Unknown)
                        TLS                                      : enabled
                        TLS mutual                               : disabled
                        Trustpoint                               : none
                        Certificate Authentication               : disabled
                        Certificate common name                  : ems.cisco.com
                        TLS v1.0                                 : enabled
                        Maximum requests                         : 128
```

```
Maximum requests per user              : 10
Maximum streams                        : 32
Maximum streams per user               : 32
Maximum concurrent streams             : 32
Memory limit (MB)                      : 1024
Keepalive time                         : 30
Keepalive timeout                      : 20
Keepalive enforcement minimum time     : 300
TLS Minimum Version                    : TLS 1.0
    TLS Maximum Version                    : TLS 1.2

    TLS cipher suites
    Default                              : none
    Default TLS1.3                       : aes_128_gcm_sha256
    : aes_256_gcm_sha384
    : chacha20_poly1305_sha256

Enable                                 : none
Disable                                : none

Operational enable                     : ecdhe-rsa-chacha20-poly1305
: ecdhe-ecdsa-chacha20-poly1305
: ecdhe-rsa-aes128-gcm-sha256
: ecdhe-ecdsa-aes128-gcm-sha256
: ecdhe-rsa-aes256-gcm-sha384
: ecdhe-ecdsa-aes256-gcm-sha384
: ecdhe-rsa-aes128-sha
: ecdhe-ecdsa-aes128-sha
: ecdhe-rsa-aes256-sha
: ecdhe-ecdsa-aes256-sha
: aes128-gcm-sha256
: aes256-gcm-sha384
: aes128-sha
: aes256-sha
Operational disable                    : none
Listen addresses                         : ANY
```

The TLS 1.3 cipher suites are not configurable, they are either fixed or static.

After completing this task, the router will use the specified TLS version for gRPC communication.

Example

For example, enabling only TLS 1.3 ensures that gRPC connections use the most secure protocol version supported by the router.

What to do next

After configuring the TLS version, verify the gRPC server status and test connectivity using a gRPC client to ensure compatibility.

Certificate common-name for dial-in using gRPC protocol

A certificate common-name for dial-in using gRPC protocol is a security configuration that:

- allows the router to generate certificates with a user-defined common-name,
- enables gRPC clients to verify the server identity using a matching hostname, and
- prevents certificate verification failures caused by fixed or mismatched common-names.

This feature enhances TLS authentication flexibility and supports secure, hostname-based validation for gRPC dial-in sessions.

Table 3: Feature History Table

Feature Name	Release Information	Description
Certificate common-name for dial-in using gRPC protocol	Release 24.1.1	You can now specify a **common-name** for the certificate generated by the router while using gRPC dial-in. Earlier, the **common-name** in the certificate was fixed as *ems.cisco.com* and was not configurable. Using a specified **common-name** avoids potential certification failures where you may specify a hostname different from the fixed common name to connect to the router. The feature introduces these changes: **CLI:** • **grpc certificate common-name** **YANG Data Model:** • New XPath for `Cisco-IOS-XR-um-grpc-cfg.yang` • New XPath for `Cisco-IOS-XR-man-ems-cfg` (see GitHub, YANG Data Models Navigator)

gRPC dial-in certificate common-name configuration

When using gRPC dial-in on Cisco IOS-XR routers, the `common-name` associated with the certificate generated by the router was previously fixed as *ems.cisco.com*, causing failures during certificate verification if a different hostname was used. From Cisco IOS XR 24.11, you can now specify the common-name in the certificate using the `grpc certificate common-name` command, allowing gRPC clients to more flexibly and securely verify the server's domain name.

Configure certificate common name for dial-in

Configure a common name to be used in EMSD certificates for gRPC dial-in.

Before you begin

Before you begin, ensure the following:

- The router is running with the correct OS image.
- gRPC is enabled and properly configured on the device.

Procedure

Step 1 Configure a common name.

Example:

```
Router#config
                        Router(config)#grpc
                        Router(config-grpc)#certificate common-name cisco.com
                        Router(config-grpc)#commit
```

Use the show command to verify the common name:

```
Router#show grpc
                        Certificate common name                : cisco.com
```

Note
For the above configuration to be successful, ensure to regenerate the certificate so that the new EMSD certificates include the configured common name.

To **regenerate** the self-signed certificate, perform the following steps.

Step 2 Remove the certificates: `/misc/config/grpc/ems.pem`, `/misc/config/grpc/ems.key`, and `/misc/config/grpc/ca.cert` from `/misc/config/grpc` file.

Example:

```
Router#run ls -ltr /misc/config/grpc/

                        total 16
                        drwx------. 2 root root 4096 Feb 14 09:17 dialout
                        -rw-rw-rw-. 1 root root 1505 Feb 14 10:58 ems.pem
                        -rw-------. 1 root root 1675 Feb 14 10:58 ems.key
                        -rw-r--r--. 1 root root 1505 Feb 14 10:58 ca.cert
Router#run rm -rf /misc/config/grpc/ems.pem /misc/config/grpc/ems.key
Router#run ls -ltr /misc/config/grpc/

                        total 8
                        drwx------. 2 root root 4096 Feb 14 09:17 dialout
                        -rw-r--r--. 1 root root 1505 Feb 14 10:58 ca.cert
```

Step 3 Restart gRPC server by toggling the TLS configuration.

Configure gRPC with non TLS and then re-configure with TLS.

Example:

```
Router#config
                        Router(config)#grpc
                        Router(config-grpc)#no-tls
                        Router(config-grpc)#commit
Router#run ls -ltr /misc/config/grpc/

                        total 8
                        drwx------. 2 root root 4096 Feb 14 09:17 dialout
                        -rw-r--r--. 1 root root 1505 Feb 14 10:58 ca.cert
Router#config
                        Router(config)#grpc
                        Router(config-grpc)#no no-tls
                        Router(config-grpc)#commit
Router#run ls -ltr /misc/config/grpc/

                        total 16
                        drwx------. 2 root root 4096 Feb 14 09:17 dialout
                        -rw-rw-rw-. 1 root root 1505 Feb 14 14:23 ems.pem
                        -rw-------. 1 root root 1675 Feb 14 14:23 ems.key
                        -rw-r--r--. 1 root root 1505 Feb 14 14:23 ca.cert
```

Copy the newly generated `/misc/config/grpc/ems.pem` certificate in this path (from the device) to the gRPC client.

The common name is successfully configured and reflected in the regenerated EMSD certificate used for gRPC dial-in.

Example

For example, after configuring `certificate common-name cisco.com` and regenerating the certificate, the output of `show grpc` displays: *Certificate common name : cisco.com*.

What to do next

After completing this task:

- Ensure the gRPC client trusts the new certificate and can establish a secure connection using the updated common name.

SPIFFE ID-based authentication and authorization services for gRPC services

A SPIFFE ID (Secure Production Identity Framework for Everyone) based authentication and authorization service is a standardized framework that:

- enables secure identification and authorization of services communicating over gRPC,
- provides interoperability for authentication and access control across diverse and distributed environments, and
- leverages SPIFFE IDs and Verifiable Identity Documents (SVIDs) to enforce mutual TLS (mTLS) and authorization policies.

SPIFFE ID-Based authentication and authorization services for gRPC services uses SPIFFE IDs and SPIFFE Verifiable Identity Documents (SVIDs) to authenticate and authorize gRPC traffic. This is especially useful in distributed systems where workloads span multiple platforms.

- Authentication: Performed via mutual TLS (mTLS) using SVIDs
- Authorization: Based on mapping SPIFFE IDs to XR usernames
- Identity format: SVIDs can be encoded as X.509 certificates or JWTs
- Integration: Enables EMS and gRPC services to enforce access control

Workflow for SPIFFE ID-based authentication and authorization for gRPC services

Mapping initialization and configuration

1. The EMS starts searching for the *spiffe-user-map.json* file at the location `/misc/config/grpc/gnsi/credentialz/spiffe-user-map.json`.
2. If the file exists, it is parsed, and the mapping is stored globally in the `aaa/auth package`.
3. If the file does not exist or parsing is unsuccessful, the mapping will be empty.
4. The EMS registers with the configuration manager to receive updates for the `aaa` configuration.

Authentication and authorization Flow

1. When processing requests in the Authentication interceptor, the spiffe-user mapping API checks for the SPIFFE ID mapping.
2. If the mapping exists, the API responds with the corresponding username.
3. If the mapping does not exist but the `aaa` configuration exists, the API responds with the configured username.
4. If neither the mapping nor the `aaa` configuration is present, the API responds with an empty string.
5. Upon a client connecting to the server, the server interceptor extracts the SPIFFE ID from the client's certificate and uses the mapping stored in the `aaa/auth package` to find the corresponding username.
6. The username identifies it and then includes the metadata into the context.
7. gRPC services that require XR Authorization will later verify the access rights for the username identified in the previous step when handling the request.
8. If the mapping is unsuccessful, the request is passed to the relevant service, such as gNMI, which then decides whether to grant or deny access based on its authorization requirements.

Authenticate and authorize gRPC service requests using the SPIFFE standard

This task describes how to authenticate and authorize gRPC service requests using the SPIFFE standard by mapping SPIFFE IDs to usernames and evaluating authorization policies.

Before you begin

Before authenticating and authorizing gRPC service requests using the SPIFFE standard, ensure the following prerequisites are met:

- Enable mutual TLS authentication with the `tls-mutual` command.
- Enable certificate authentication with the `certificate-authentication` command to facilitate SPIFFE ID recognition. For more information, see Configure authentication for gRPC services, on page 2.
- Configure the gNSI Authz policy by setting the principal to the SPIFFE-ID for service-level authorization (gNSI AuthZ).

After establishing the connection, the gRPC server extracts the SPIFFE ID from the client's certificate.

To authenticate and authorize gRPC service requests using the SPIFFE standard, follow these steps:

Procedure

Step 1 Configure the username in the system.

Example:

```
Router#show running-config aaa
                        Thu Oct 12 11:43:15.771 UTC
                        username cisco
                        group root-lr
                        group cisco-support
                        password 7 104D000A061843595F
                        !
```

Step 2 Map the SPIFFE ID to a username using the **aaa map-to username** command. This command assigns a default username to any SPIFFE ID.

```
Router(config)#aaa map-to username cisco spiffe-id any
                           Router(config)#commit
```

Note
Each SPIFFE ID supports only one username.

Step 3 Evaluate the client's SPIFFE ID against the service-level authorization policy (gNSI AuthZ). For more information about gNSI authz policies.

The gRPC service request is authenticated and authorized using the SPIFFE ID mapped to a system username and evaluated against the gNSI AuthZ policy.

Example

For example, after mapping the SPIFFE ID to the username `cisco`, the system uses this identity to authorize access based on the configured gNSI AuthZ policy.

What to do next

After completing this task:

- Monitor gRPC logs to verify successful authentication and authorization events using SPIFFE IDs.

gRPC network security interface

The gRPC Network Security Interface (gNSI) is a repository that contains essential security infrastructure services for the safe operation of an OpenConfig platform.

- **Authorization protocol buffer:** Defines gNSI.authz policies to restrict access to gRPC services based on client permissions.
- **Policy enforcement:** Enables configuration of RPC service access on routers and restricts unauthorized updates to sensitive RPCs.
- **Fallback behavior:** In the absence of a valid policy, the system defaults to zero-policy mode, allowing access to all services for configured users.
- **Secure ZTP integration:** Default gRPC-level authorization policies can be provisioned using Secure Zero Touch Provisioning (ZTP).
- **CLI support:** Includes commands to load and view authorization policies on supported platforms.

Table 4: Feature History Table

Feature Name	Release Information	Feature Description
Release 7.11.1	This release implements authorization mechanisms to restrict access to gRPC applications and services based on client permissions. This is made possible by introducing an authorization protocol buffer service for gRPC Network Security Interface (gNSI). Prior to this release, the gRPC services in the gNSI systems could be accessed by unauthorized users. This feature introduces the following change: **CLI**: To view the specification of gNSI, see Github repository.	

The gRPC network security interface (gNSI) contains security infrastructure services for the OpenConfig platform operation. These services manage a network device's certificates and authorization policies, including an authorization protocol buffer.

gNSI authorization protocol buffer under gRPC contains gNSI.authz policies to prevent unauthorized access to sensitive information and defines an API for configuring RPC services on a router, controlling user access, and restricting authorization to update specific RPCs

gRPC default and zero-policy mode authorization

By default, gRPC-level authorization policy is provisioned using Secure ZTP. In zero-policy mode, or when no policy is present, gRPC authorization policy configuration can be used to restrict access to specific users. The default policy allows access to all RPCs, except for gNSI.authz RPCs.

Policy fallback and management commands

In the absence of a specified policy or if the policy is invalid, the router defaults to zero-policy mode, granting access to all gRPC services for users with configured profiles. To revert an invalid policy, use the exec command **gnsi load service authorization policy.** For creating user profiles and updating authorization policies. The **show gnsi service authorization policy** command displays the active policy on a router.

We have introduced these commands in this release :

- **gnsi load service authorization policy**: To load and update the gRPC-level authorization policy in a router.
- **show gnsi service authorization policy**: To view the active policy in a router.

Precedence of gNSI over gNOI

When both gNSI and gNOI are configured, gNSI takes precedence over gNOI. If niether gNSI nor gNOI is configured, then tls trsutpoint's data is considered for certificate management.

gNSI RPCs

These RPCs are used to perform key operations at the system level such as updating and displaying the current status of the authorization policy in a router.

The table lists the gNSI RPC operations and their description.

Table 5: Operations

gNSI RPC Operation	Description
gNSI.authz.Rotate()	Updates the gRPC-level authorization policy.
gNSI.authz.Probe()	Verifies the authenticity of a user based on the defined policy of the gRPC-level authorization policy engine.
gNSI.authz.Get()	Shows the current instance of the gRPC-level authorization policy, including the version and date of creation of the policy.

gRPC-level authorization policies

gRPC-level authorization policies are configuration rules that control access to gRPC services on the router based on user identity and permissions.

- Defined during secure ZTP deployment
- Can be updated post-deployment using CLI or configuration
- Affect access to gRPC services like gNMI and gNSI

By default, the gRPC-level authorization policy is configured at the time of router deployment using secure ZTP. You can update the same gRPC-level authorization policy using any of these two methods:

- gRPC-level authorization policy update using gNSI client, on page 15
- gRPC-level authorization policy update using Exec command, on page 18

gRPC-level authorization policy update using gNSI client

You can update gRPC-level authorization policy using gNSI authz streaming RPCs.

Before you begin

When a router boots for the first time, it should have these prerequisites:

- The gNSI.authz service is up and running.
- The default gRPC-level authorization policy is added for all gRPC services.
- The default gRPC-level authorization policy allows access to all RPCs.

SUMMARY STEPS

1. Initiate the **gNSI.authz.Rotate()** streaming RPC. This step creates a streaming connection between the router and management application (client).
2. Upload new gRPC-level authorization policy using the **UploadRequest** message.
3. The router activates the gRPC-level authorization policy.
4. The router sends the `UploadResponse` message back to the client after activating the new policy.
5. The client verifies the new gRPC-level authorization policy using separate **gNSI.authz.Probe()** RPCs.
6. The client sends the **FinalizeRequest** message, indicating the previous gRPC-level authorization policy is replaced.
7. Example

DETAILED STEPS

Procedure

	Command or Action	Purpose
Step 1	Initiate the **gNSI.authz.Rotate()** streaming RPC. This step creates a streaming connection between the router and management application (client).	**Note** Only one `gNSI.authz.Rotate()` must be in progress at a time. Any other RPC request is rejected by the server.

	Command or Action	Purpose
Step 2	Upload new gRPC-level authorization policy using the **UploadRequest** message.	**Note** • There must be only one gRPC-level authorization policy in the router. All the policies must be defined in the same gRPC-level authorization policy which is being updated. As `gNSI.authz.Rotate()` method replaces all previously defined or used policies once the **finalize** message is sent. • The upgrade information is passed to the version and the created_on fields. These information are not used by the gNSI.authz service. It is designed to help you to track the active gRPC-level authorization policy on a particular router.
Step 3	The router activates the gRPC-level authorization policy.	
Step 4	The router sends the `UploadResponse` message back to the client after activating the new policy.	
Step 5	The client verifies the new gRPC-level authorization policy using separate **gNSI.authz.Probe()** RPCs.	
Step 6	The client sends the **FinalizeRequest** message, indicating the previous gRPC-level authorization policy is replaced.	**Note** It is not recommended to close the stream without sending the **finalize** message. It results in the abandoning of the uploaded policy and rollback to the one that was active before the `gNSI.authz.Rotate()` RPC started.

Step 7	Example
	Example:
	This gRPC-level authorization policy grants admins, V1, V2, V3, and V4 access to all RPCs defined by the gNSI.ssh interface, while denying access to all other users.
	`{`
	`  "version": "version-1",`
	`  "created_on": "1632779276520673693",`
	`  "policy": {`
	`  "name": "gNSI.ssh policy",`
	`  "allow_rules": [{`
	`  "name": "admin-access",`
	`  "source": {`
	`  "principals": [`
	`  "spiffe://company.com/sa/V1",`
	`  "spiffe://company.com/sa/V2"`
	`  ]`
	`  },`
	`  "request": {`
	`  "paths": [`
	`  "/gnsi.ssh.Ssh/*"`
	`  ]`
	`  }`
	`  }],`
	`  "deny_rules": [{`	

	Command or Action	Purpose

	<pre> "name": "sales-access",
	      "source": {
	        "principals": [
	          "spiffe://company.com/sa/V3",
	          "spiffe://company.com/sa/V4"
	        ]
	      },
	      "request": {
	        "paths": [
	         "/gnsi.ssh.Ssh/MutateAccountCredentials",
	
	          "/gnsi.ssh.Ssh/MutateHostCredentials"
	        ]
	      }
	    }]
	  }
	}</pre> **Example:** <pre>{
	  "version": "version-1",
	  "created_on": "1632779276520673693",
	  "policy": {
	    "name": "gNSI.ssh policy",
	    "allow_rules": [{
	      "name": "admin-access",
	      "source": {
	        "principals": [
	          "spiffe://company.com/sa/V1",
	          "spiffe://company.com/sa/V2"
	        ]
	      },
	      "request": {
	        "paths": [
	          "/gnsi.ssh.Ssh/*"
	        ]
	      }
	    }],
	    "deny_rules": [{
	      "name": "sales-access",
	      "source": {
	        "principals": [
	          "spiffe://company.com/sa/V3",
	          "spiffe://company.com/sa/V4"
	        ]
	      },
	      "request": {
	        "paths": [
	         "/gnsi.ssh.Ssh/MutateAccountCredentials",
	
	          "/gnsi.ssh.Ssh/MutateHostCredentials"
	        ]
	      }
	    }]
	  }
	}</pre>	

After completing this task, the updated gRPC-level authorization policy is applied to the router configuration.

Example

For example, you can verify the updated policy using the 'show grpc authorization policy' command.

What to do next

After updating the policy, monitor gRPC access logs to ensure the new rules are enforced as expected.

gRPC-level authorization policy update using Exec command

Use this task to update the default gRPC-level authorization policy configured during secure ZTP deployment using the exec command.

Before you begin

Ensure you have access to the router CLI with sufficient privileges to execute configuration commands.

SUMMARY STEPS

1. Create user profiles for those to be added to the authorization policy.
2. Enable **tls-mutual** to establish the secure mutual between the client and the router.
3. Define the gRPC-level authorization policy
4. Copy the gRPC-level authorization policy to the router
5. Activate the gRPC-level authorization policy to the router.
6. Verification

DETAILED STEPS

Procedure

	Command or Action	Purpose

Step 1	Create user profiles for those to be added to the authorization policy.
	Example:
	`Router(config)#username V1`
	`Router(config-un)#group root-lr`
	`Router(config-un)#group cisco-support`
	`Router(config-un)#secret x`
	`Router(config-un)#exit`
	`Router(config)#username V2`
	`Router(config-un)#group root-lr`
	`Router(config-un)#password x`
	`Router(config-un)#exit`
		You can skip this step if you have already defined the user profiles.

	Command or Action	Purpose
	Router(config)#**username V3** Router(config-un)#**group root-lr** Router(config-un)#**password x** Router(config-un)#**commit**	
Step 2	Enable **tls-mutual** to establish the secure mutual between the client and the router. **Example:** Router(config)#**grpc** Router(config-grpc)#**port 0** Router(config-grpc)#**tls-mutual** Router(config-grpc)#**certificate-authentication** Router(config-grpc)#**commit**	

Step 3	Define the gRPC-level authorization policy
	Example:
	{
	"name": "authz",
	"allow_rules": [
	{
	"name": "allow all gNMI for all users",
	"source": {
	"principals": [
	"*"
	]
	},
	"request": {
	"paths": [
	"*"
	]
	}
	}
	],
	"deny_rules": [
	{
	"name": "deny gNMI set for oper users",
	"source": {
	"principals": [
	"V1"
	]
	},
	"request": {
	"paths": [
	"/gnmi.gNMI/Get".
	]
	}
	},
	{	

	Command or Action	Purpose

	"name": "deny gNMI set for oper users",
	"source": {
	"principals": [
	"V2"
	]
	},
	"request": {
	"paths": [
	"/gnmi.gNMI/Get"
	]
	}
	},
	{
	"name": "deny gNMI set for oper users",
	"source": {
	"principals": [
	"V3"
	]
	},
	"request": {
	"paths": [
	"/gnmi.gNMI/Set"
	]
	}
	}
	]
	}	

Step 4	Copy the gRPC-level authorization policy to the router **Example:** -bash-4.2$ **scp test.json V1@192.0.2.255:/disk0:/** Password: test.json 100% 993 161.4KB/s 00:00 -bash-4.2$	
Step 5	Activate the gRPC-level authorization policy to the router. **Example:** Router(config)#**gnsi load service authorization policy /disk0:/test.json** Successfully loaded policy	
Step 6	Verification **Example:** Router#**show gnsi service authorization policy** Wed Jul 19 10:56:14.509 UTC{ "version": "1.0",	In this example, User1 user tries to access the **get** RPC request for which the permission is denied in the above authorization policy. bash-4.2$ **./gnmi_cli -address 198.51.100.255 -ca_crt certs/certs/ca.cert**

	Command or Action	Purpose

	<pre>"created_on": 1700816204,
	"policy": {
	"name": "authz",
	"allow_rules": [
	{
	"name": "allow all gNMI for all users",
	"request": {
	"paths": [
	"*"
	]
	},
	"source": {
	"principals": [
	"*"
	]
	}
	}
	],
	"deny_rules": [
	{
	"name": "deny gNMI set for oper users",
	"request": {
	"paths": [
	"/gnmi.gNMI/*"
	]
	},
	"source": {
	"principals": [
	"User1"
	]
	}
	}
	]
	}
	}</pre>
		-client_crt certs/certs/User1.pem -client_key certs/certs/User1.key -server_name ems.cisco.com -get -proto get-oper.proto
		Output
		<pre>E0720 14:49:42.277504 26473 gnmi_cli.go:195] target returned RPC error for Get("path:{origin:"openconfig-interfaces" elem:{name:"interfaces"} elem:{name:"interface" key:{key:"name" value:"HundredGigE0/0/0/0"}}} type:OPERATIONAL encoding:JSON_IETF"):</pre>**rpc error: code = PermissionDenied desc = unauthorized RPC request rejected**

After completing this task, the gRPC-level authorization policy is updated and active on the router.

Example

For example, after executing the update command, you can verify the policy using the 'show grpc authorization policy' command.

What to do next

After updating the policy, validate the configuration by checking the gRPC service behavior and authorization logs.

gNSI Credentialz updates

A gNSI Credentialz update is a process in a network using gNSI that allows you to change or update SSH credentials while the router is operational. These credentials include:

- Passwords

- Keys
- Certificates

Table 6: Feature History Table

Feature Name	Release Information	Description
gNSI Credentialz Update	Release 24.2.11	To improve communication confidentiality and security, you can now update or rotate account-specific and host-specific SSH credentials on a router. You can access the latest SSH credentials through the gNSI credentialz RPC. The updated SSH credentials encompass passwords, host keys, and certificates. To view the specification of gNSI credentialz RPCs and messages, see the GitHub repository.

This update mechanism improves communication confidentiality and security by enabling real-time credential rotation without requiring a router reboot.

- **Live credential rotation:** Credentials can be updated while the router is running.
- **Account and host-specific updates:** Supports both user account and host-level SSH credential changes.
- **gNSI credentialz RPC:** Provides the interface to perform credential updates securely.

gNSI Rotate Credentialz RPC

Starting from Release 24.2.1, Cisco IOS XR supports four gNSI RPCs to rotate SSH credentials securely. These RPCs allow administrators to manage authentication policies and key material on network devices.

- **RotateAccountCredentials:** Replaces the existing SSH authentication service policy with a new one if valid.
- **RotateHostParameters:** Updates the Certificate Authority (CA) public key and the SSH server's key and certificate.
- **CanGenerateKey:** Checks whether the target device can generate a public/private key pair.
- **GetPublicKeys:** Retrieves the current public keys configured on the host.

gNSI Rotate Credentialz RPC support in Cisco IOS XR

Table 7: gNSI Rotate Credentialz RPC support in Cisco IOS XR

gNSI Rotate Credentialz RPC	Run This When	For More Information
RotateAccountCredentials	You want to specify an SSH authentication service policy for the network element. If the policy is valid, it replaces the existing policy.	See RotateAccountCredentials

gNSI Rotate Credentialz RPC	Run This When	For More Information
RotateHostParameters	You want to change both the Certificate Authority (CA) public key and the key and certificate used by the SSH server.	See RotateHostParameters
CanGenerateKey	You want to check whether the target can generate a public or private key pair.	See CanGenerateKey
GetPublicKeys	You want to get the current public keys from the host. It returns each configured key in the provided list.	See GetPublicKey

Rotate account credentials

Credential rotation using gNSI is a security automation feature that updates user credentials on routers through RPC-based operations.

- Updates user-specific authorized keys and authorized principals for SSH access.
- Invalidates old credentials and logs credential rotation activities.
- Notifies stakeholders and enforces secure access policies across the network.

This enhances overall network security by ensuring credentials are rotated regularly and securely.

Prerequisites

- Configure a user account on your router.
- Configure SSH Version 2.

The table outlines the messages that `Rotate Account Credentials` RPC supports, along with their descriptions.

Table 8: Rotate Account Credentials RPC Messages

Message	Description
AuthorizedKeysRequest	This message defines the authorized key list for password-less SSH accepted by the router's SSH service. The gNSI client dispatches an `AuthorizedKeysRequest` to the router to update or replace credentials on the SSH service. The router responds with a `AuthorizedKeysResponse` message to the gNSI client. It supports these keys: • RSA 2048, RSA 4096 bits • ECDSA-p-256, ECDSA-p-521 • Ed25519

Message	Description
AuthorizedUsersRequest	This message performs a user authorization check. User authorization can be done using both static and dynamic methods. Static Authorization: You can perform static authorization based on a principal name (unique identifier for a user) using Cisco SSH. For static authorization, use the `AuthorizedUsersRequest` message. Dynamic Authorization: For dynamic authorization, use the `AuthorizedPrincipalCheckRequest` message. For details, see Rotate host parameters, on page 24. CiscoSSH supports the user authorization using AuthorizedPrincipalsFile. AuthorizedPrincipalsFile contains pairs of account names and their corresponding principal names that the router recognizes for certificate-based authentication. For more details, see AuthorizedPrincipalsFile.

Rotate host parameters

Rotate host parameters is a gNSI RPC-based mechanism that manages SSH host credentials and authentication policies on routers.

- Updates and verifies host account credentials to maintain secure SSH access.
- Automatically reverts to old credentials if validation fails to prevent lockouts.
- Supports dynamic and static authorization using OpenSSH and CiscoSSH mechanisms.

This ensures secure, automated credential rotation and policy enforcement across network devices.

Prerequisites

- Configure a user account on your router.
- Configure SSH Version 2.

The table outlines the messages that `Rotate Host Parameters` RPC supports, along with their descriptions.

Table 9: Rotate host parameters RPC support in Cisco IOS XR

Message	Description
CA public key	The SSH server uses the `CA public key` message to verify the gNSI client certificates presented during connection establishment. Without Host Identity Based Authorization (HIBA), these keys are supported: • RSA 2048, RSA 4096 bits • ECDSA-p-256, ECDSA-p-521 • Ed25519

Message	Description

Server keys	The `Server keys` message includes host keys and router certificates that serve as credentialz for the gNSI client.
	If the host keys are generated externally, they must be specified in the `Server keys` request.
	It supports these keys:
	• RSA 2048, RSA 4096 bits
	• ECDSA-p-256, ECDSA-p-521
	• Ed25519
	It supports these router certificates:
	• Router certificates with HIBA Support
	  • ssh-rsa-cert-v01@openssh.com
	• Router certificates without HIBA support:
	  • ecdsa-sha2-nistp256-cert-v01@openssh.com
	  • ecdsa-sha2-nistp521-cert-v01@openssh.com
	  • ssh-ed25519-cert-v01@openssh.com
	  • rsa-sha2-256-cert-v01@openssh.com
	  • rsa-sha2-512-cert-v01@openssh.com

Generate key	The `Generate Key` message is used for host key management in SSH. When the host keys are generated by the router, this message triggers the creation of new host keys for SSH host key management. The `Generate key` message supports these keys: • RSA 2048, RSA 4096 bits • ECDSA-p-256, ECDSA-p-521 • Ed25519

Message	Description

AllowedAuthenticationRequest	The `AllowedAuthenticationRequest` message specifies the permissible authentication methods for the gNSI client authentication.
	The supported authentication methods are as follows:
	• Keyboard interactive
	• Password-based
	• Pubkey-based
	  • OpenSSH certificate-based
	  • Public key-based
	By default, the SSH server allows all authentication methods.

AuthorizedPrincipalCheckRequest	The `AuthorizedPrincipalCheckRequest` message supports the dynamic authorization of the user against the principal name using the OpenSSH or CiscoSSH. Setting the TOOL_HIBA_DEFAULT flag prompts the router to use the HIBA binary for dynamic authorization. Un setting the HIBA_DEFAULT flag switches the router to use a static authorization. Dynamic Authorization: You can enforce the user for authorization check using HIBA. **Note** The support is only for ssh-rsa-cert-v01@openssh.com CiscoSSH supports AuthorizedPrincipalCheck using `AuthorizedPrincipalsCommand` and `AuthorizedPrincipalsCommandUser` AuthorizedPrincipalsCommand: This command generates the list of allowed certificate principals by executing a HIBA binary (By setting the TOOL_HIBA_DEFAULT flag). AuthorizedPrincipalsCommandUser: This command specifies the user account under which the system executes the `AuthorizedPrincipalsCommand`. For more details on the specification, see AuthorizedPrincipalsCommandUser

CanGenerateKey

The `CanGenerateKey` RPC is a capability check that determines whether a router can generate a public or private key pair.

- RSA 2048, RSA 4096 bits
- ECDSA-p-256, ECDSA-p-521
- Ed25519

GetPublicKey

The `GetPublicKey` RPC gets the available public keys from the router and displays them. It supports these keys:

- RSA 2048, RSA 4096 bits
- ECDSA-p-256, ECDSA-p-521
- Ed25519

Manage certificates using Certz.proto

Manage certificates using Certz.proto is a gRPC-based certificate management framework that:

- Provides a single Rotate RPC to upload certificates, keys, CA bundles, and CRLs.
- Supports SSL profile creation, deletion, and listing through dedicated RPCs.
- Enables secure and flexible certificate lifecycle operations using standard cryptographic algorithms.

This approach simplifies certificate handling and enhances security by associating all certificate entities with unique SSL profiles.

Table 10: Feature History Table

Feature Name	Release Information	Feature Description
Manage Certificates using Certz.proto	Release 24.1.1	Instead of using multiple RPCs, Certz.proto provides a bidirectional Rotate RPC to replace, revoke, or load a certificate. It also provides additional APIs to install Public Key Infrastructure (PKI) entities such as like identity certificates, trust-bundles, and Certificate Revocation Lists (CRLs) for a gRPC Server. This feature introduces the following changes: CLI: • **grpc gnsi service certz ssl-profile-id** • **show grpc certificate** Yang Data Models: • Cisco-IOS-XR-man-ems-cfg.yang (see Github, YANG Data Models Navigator)

The Manage certificates using Certz.proto handles certificate operations on target devices, utilizing Certz RPCs. The `certz.proto` file is available in the Github repository.

Certzs certificate identification and management

cert.proto: A certificate identifier differentiates between leaf certificates. However, the certificate authority (CA) bundle lacks an identifier, meaning a new request to load a bundle could overwrite the existing one. Separate RPCs are used to replace, load, and revoke a certificate.

certz.proto: Entities like Certificate, CA bundle, key, CRL, and authentication policy to a unique SSL profile.

A single `Rotate()` RPC uploads all entities at once, including the certificate, the key, the CA bundle, and the CRL.

These are the supported certz.proto cryptographic algorithms:

- Rivest-Shamir-Adleman (RSA)
- Elliptic Curve Digital Signature Algorithm (ECDSA)
- ED25519, a public-key signature system

Certz.proto offers unified certificate management, unique SSL profile association, and support for multiple cryptographic algorithms, making it a comprehensive solution for managing SSL profiles.

Note If neither cert.proto nor certz.proto is configured, then tls trustpoint data is considered for certificate management.

The table describes the RPCs supported under Certz.proto.

Table 11: Certz RPCs

RPC	Description
AddProfile	`AddProfile` is part of SSL profile management. It allows adding a new SSL profile. When an SSL profile is added, all its elements, that is, certificate, CA trusted bundle and a set of certificate revocation lists are NULL or Empty. So, before an SSL profile can be used these entities have to be 'rotated' using the `Rotate()` RPC. **Note** An attempt to add an already existing profile is rejected with an error.
Rotate	`Rotate` function replaces or adds an existing device certificate, CA certificates (trust bundle), or a certificate revocation list (CRL) bundle on the target.. The new device certificate can be created from a target-generated or client-generated CSR (Certificate Signing Request). In the latter case, the client must provide the corresponding private key with the signed certificate.

RPC	Description
DeleteProfile	`DeleteProfile` is part of SSL profile management. It allows for removing an existing SSL profile. **Note** An attempt to delete a not existing profile results in an error. The profile used by the gRPC server can't be deleted and an attempt to remove it will be rejected with an error.
GetProfileList	GetProfileList is part of SSL profile management. It allows for retrieving a list of IDs of SSL profiles present on the target.
CanGenerateCSR	An RPC to ask a target if it can generate a CSR.

SSL profile

An SSL profile is a named set of SSL settings that determine how end-user systems connect to or from SSL-based applications or interfaces. The settings in an SSL profile include information about the version of SSL or TLS to be used, certificates, keys, and other parameters related to SSL or TLS communication. By using profiles, administrators can manage and apply these settings more easily across multiple applications or connections.

SSL profile characteristics

- **Logical Grouping**: SSL profiles logically group certificates, private keys, Certificate Authority (CA) chain of certificates (a.k.a. a CA trust bundle), and a list of Certificate Revocation Lists (CRLs) into a single set that can be assigned to a gRPC server.
- **Default Profile**: There's at least one profile present on a target—the one used by the gRPC server. Its ID is `gNxI`. When the `ssl_profile_id` field in the `RotateCertificateRequest` message isn't set (or is set to an empty string), it refers to this SSL profile by default.
- **Non-Removable Profile**: You can't remove the gRPC SSL profile (`gNxI`).

Configure gNSI Certz

This task guides you through configuring gNSI Certz by creating, rotating, and activating SSL profiles, and verifying the certificate configuration using gRPC.

Before you begin

Before you begin, ensure the following:

- Ensure you've created and stored SSL-Profile at `cd/misc/config/grpc/gnsi/certz/ssl_profiles/`

Procedure

Step 1 Create SSL-Profile using AddProfile RPC.

Step 2 Rotate SSL-profile using Rotate RPC. You can't rotate SSL-profile using a command line interface.

Step 3 Activate the profile using **grpc gnsi service certz ssl-profile-id.**

Example:

Router (config-grpc) #**gnsi service certz profile ssl-profile id <ssl-profile-name>**

Step 4 Verify that certz.proto is configured using the **show grpc certificate**.

Example:

```
Router#show grpc certificate
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number: 32 (0x20)
        Signature Algorithm: sha256WithRSAEncryption
        Issuer: CN=localhost,O=OpenConfig,C=US
        Validity
            Not Before: Nov  8 08:49:38 2023 GMT
            Not After : Mar 22 08:49:38 2025 GMT
        Subject: CN=ems,O=OpenConfig,C=US
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                RSA Public-Key: (4096 bit)
                Modulus:
                    00:ea:6a:6c:25:be:9f:15:71:ce:74:89:03:ec:ef:
                    0b:3b:de:58:a8:7e:28:b8:cf:b3:82:91:b4:5c:42:
                    e7:d8:28:98:35:bd:35:60:a7:4e:f8:77:02:46:5f:
                    27:a4:16:cf:3c:e3:24:28:69:9c:22:1e:e3:52:96:
                    71:87:7c:40:0c:1f:dd:30:ea:dc:40:ca:93:00:54:
                    5e:de:20:54:5b:f4:2f:9f:19:6f:71:61:28:69:3d:
                    97:26:ab:e1:5f:53:3c:f1:a2:c3:14:f4:01:90:1a:
                    e3:08:7b:51:c9:5d:aa:6d:eb:99:a4:08:97:d3:72:
                    8c:86:a3:f3:b3:77:10:72:e7:a9:3b:fc:38:65:3d:
                    41:1a:f5:cf:3e:a0:d8:17:d6:d5:53:86:49:a3:dc:
                    cc:3a:d9:6d:46:25:b0:f9:3b:98:fa:2f:98:09:08:
                    51:ac:2c:b1:43:c4:b7:96:3e:4e:4e:a6:a5:36:1f:
                    1f:0f:6a:6a:1a:ea:72:6e:74:90:21:05:fb:26:df:
                    81:0d:96:e7:13:94:62:2b:ce:3c:7c:de:32:f4:d9:
                    fa:24:ce:f5:b2:0f:d3:f7:4b:6b:ee:bd:cf:ac:a6:
                    ed:69:37:fc:d3:4f:3b:46:8b:1b:62:4d:3b:60:30:
                    74:68:50:4e:48:35:5f:15:66:9a:01:7c:37:1f:e1:
                    5a:8a:d9:c0:2c:3e:12:fd:71:30:13:b8:b7:16:98:
                    03:27:6d:45:c4:0f:34:fd:f1:aa:29:8e:c1:63:ac:
                    57:04:f6:a7:83:83:06:45:dc:0f:f9:de:f9:1e:b6:
                    d8:5a:bc:3a:98:f8:ac:b0:be:3f:87:df:8c:5e:47:
                    12:ca:77:70:26:14:02:14:79:fa:6f:1f:ab:ee:06:
                    2c:83:93:e4:22:db:37:83:90:c1:72:5b:36:78:1b:
                    6d:0a:06:72:76:dc:89:df:86:89:43:54:03:55:bd:
                    fc:a0:9a:d6:8e:5d:22:87:a2:32:19:35:c8:17:4e:
                    1c:1b:5e:81:9d:a5:67:9e:a7:ed:06:e8:e2:91:f1:
                    ae:f9:19:b1:ae:a8:e6:66:14:2c:6d:a6:c3:0f:8b:
                    7f:ef:c0:60:cb:c2:52:a5:46:1e:a4:20:52:f8:93:
                    93:2b:02:23:98:90:81:b3:e6:c4:4e:8f:85:a6:ff:
                    4e:8e:dd:6c:12:ea:db:58:7f:3c:66:c4:38:96:44:
                    d1:5b:da:c2:66:6a:4e:97:4d:99:59:9f:24:a0:4a:
                    57:b6:9d:69:22:f7:5a:10:cb:96:bc:58:ca:96:0e:
                    ab:b0:4d:14:da:03:e1:d3:24:c1:f2:bd:40:32:20:
                    82:66:4d:78:4b:13:c6:bd:66:a9:83:2f:15:29:7e:
                    11:95:37
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Key Usage: critical
                Digital Signature
            X509v3 Extended Key Usage:
                TLS Web Client Authentication, TLS Web Server Authentication
```

```
            X509v3 Authority Key Identifier:
                keyid:0A:A8:9A:6A:23:34:AE:CA:96:00:2C:F3:04:38:14:E3:D4:8D:77:BD

            X509v3 Subject Alternative Name:
                DNS, IP Address:64.103.223.56
    Signature Algorithm: sha256WithRSAEncryption
         b9:89:ec:60:3d:8d:7d:9c:dc:08:56:89:99:44:92:98:45:b6:
         97:ba:e3:e5:f2:48:b2:44:8d:db:23:bb:a1:c0:62:79:78:18:
         d7:55:f6:4a:67:5b:75:e0:c0:0b:52:51:07:36:d5:6c:c7:67:
         48:86:8d:dd:70:1c:9f:7c:a1:7b:aa:a5:4e:e1:ad:cf:4c:e5:
         81:db:92:cf:88:70:5a:1c:8d:de:0d:e8:b3:05:de:b9:04:4d:
         23:e1:de:66:e5:08:bd:2e:31:0a:07:a6:c0:00:3a:38:2f:00:
         cd:cf:be:e2:1f:12:9f:8a:44:8d:2d:24:d5:d3:bb:9e:db:70:
         bf:89:ea:0c:31:b4:b2:fc:3d:73:f5:17:09:07:54:ab:2f:23:
         cb:66:0e:0e:7a:9e:21:bf:1e:bf:07:f1:fc:09:88:23:4e:2d:
         5d:08:35:16:cd:07:df:25:34:7f:42:0a:dc:6f:d0:ec:9d:99:
         72:d8:5f:d6:7e:6f:cc:67:4d:d7:b9:b8:c8:56:75:db:56:1e:
         03:1b:6d:37:21:4d:e0:f1:e2:80:99:40:24:24:f2:e4:9b:7e:
         6c:bc:f7:f9:3a:b6:fc:8e:dd:9a:cd:dd:88:15:d7:46:71:d2:
         11:20:86:8f:ea:c5:a8:e8:4e:b6:ef:9b:06:5b:b1:c4:11:36:
         38:7a:63:8e:1a:a6:a8:f8:bb:7d:0b:a6:f2:89:49:94:ac:0c:
         8b:c4:fc:02:e8:b2:b8:27:bc:70:95:32:83:09:f5:de:68:34:
         3f:a4:5a:73:dc:92:15:2c:0e:ab:46:dd:13:06:98:aa:08:2d:
         b8:37:a0:52:4b:ba:f7:be:ed:68:cd:fb:67:3b:66:ea:16:85:
         61:75:cf:06:85:a0:06:e8:4a:3e:63:72:c1:79:c7:fd:d4:85:
         74:d8:ea:66:d3:42:74:e2:fb:7c:9e:93:4b:24:2f:ad:c5:13:
         bc:eb:83:f7:6d:3e:53:9a:ec:16:85:b7:b5:6c:77:48:53:7e:
         19:2e:48:2d:83:35:7b:b9:66:5e:12:b4:f3:ee:e8:b2:3b:ba:
         18:46:91:b0:f9:6f:b0:d5:17:a8:de:5c:a0:0e:35:85:7b:c0:
         e3:79:06:fa:ad:8e:f2:28:ab:09:19:b7:f0:f3:9e:cb:94:93:
         b7:04:63:74:82:c3:71:3b:16:8b:58:c7:fa:ff:ff:2a:97:91:
         e7:1d:06:ab:0a:6c:cc:a0:41:31:54:f2:e7:db:a3:b5:22:c4:
         ab:ec:e2:5d:86:e6:ac:a5:c6:e2:0e:15:44:a2:32:42:3d:07:
         65:0a:0d:58:2e:22:3c:7b:e3:e8:8e:2e:60:47:f0:60:04:89:
         64:65:fc:fc:74:dd:4d:7f
```

The gNSI Certz SSL profile is successfully configured, activated, and verified using gRPC certificate inspection.

Example

For example, after activating the profile with `gnsi service certz profile ssl-profile id <ssl-profile-name>`, the `show grpc certificate` command displays the certificate details including issuer, subject, and validity period.

What to do next

After completing this task:

- Ensure the gNSI client trusts the activated certificate and can securely communicate with the router.

gNSI EnrollZ and AttestZ

gNSI EnrollZ and AttestZ are TPM-based services that enhance device security through cryptographic identity verification and boot-time integrity checks.

- EnrollZ handles TPM 2.0 enrollment and provisioning of TLS and attestation certificates.

- AttestZ performs TPM 2.0 attestation by validating Platform Configuration Register (PCR) values during boot.
- These services eliminate the need for vendor Certificate Authorities and align with Trusted Computing Group (TCG) specifications.

They ensure that only verified devices receive sensitive credentials and are fully controlled by the device owner.

Table 12: Feature History Table

The EnrollZ service handles the TPM 2.0 enrollment workflow, involving cryptographic verification of the device's TPM-rooted identity and provisioning of attestation and Transport Layer Security (TLS) certificates by the device owner. This ensures that the device is under the control of the owner and not dependent on external vendor Certificate Authorities (CAs) during the attestation process.

The AttestZ service manages the TPM 2.0 attestation workflow, confirming the device's integrity throughout the boot process by comparing observed Platform Configuration Register (PCR) values against expected ones to verify the device's boot state.

This approach simplifies the TPM enrollment process for device owners, enhances control over certificate management, and eliminates external dependencies, while aligning with Trusted Computing Group (TCG) specifications.

Enroll a TPM 2.0 on network devices

The Trusted Platform Module (TPM) 2.0 enrollment workflow is a secure process for network devices to obtain the necessary credentials and configurations for TPM management.

This workflow is initiated after the device boot process and involves interaction with various gRPC API endpoints.

Before you begin

- Device has completed the Bootz workflow.
- Device is equipped with a default SSL profile using the Secure Unique Device Identifier (SUDI) key pair and certificate.
- EnrollZ service is available and ready to enroll the TPM on the control card.
- Router owner has access to the trust bundle/anchor from the router vendor.

Procedure

Step 1 Prepare device for TPM enrollment: Ensure the device has completed the Bootz workflow and is ready to serve TPM enrollment gRPC API endpoints on the required port.

Step 2 Trigger EnrollZ service: Use the `GetIakCert` API to retrieve the Initial Attestation Key (IAK) and IDevID certificates.

Step 3 Verify and validate certificates:

- Verify the signature over the IAK certificate using the trust bundle/anchor from the router vendor.
  - Root Certificate: Cisco ECC Root CA

```
-----BEGIN CERTIFICATE-----
MIIByDCCAU6gAwIBAgIBAzAKBggqhkjOPQQDAzAsMQ4wDAYDVQQKEwVDaXNjbzEa
MBgGA1UEAxMRQ2lzY28gRUNDIFJvb3QgQ0EwIBcNMTMwNDA0MDgxNTQ0WhgPMjA5
OTA5MDcxNjI0MDdaMCwxDjAMBgNVBAoTBUNpc2NvMRowGAYDVQQDExFDaXNjbyBF
Q0MgUm9vdCBDQTB2MBAGByqGSM49AgEGBSuBBAAiA2IABH7Aw7zYG8bzZ5FN1niM
5rV1OQR/L/5g0Kx3KtNtAkFFVGewWLcZv8y9SzZce1uDOOym100wJmGzS4urXg4V
JjtMiKsCcMkiNwJQdcDV1Eg0x79YU/6uy49zIPUGWxKHyqNCMEAwDgYDVR0PAQH/
BAQDAgEGMA8GA1UdEwEB/wQFMAMBAf8wHQYDVR0OBBYEFKRFti+jMbF2FbAKGDPK
9q1PPSgEMAoGCCqGSM49BAMDA2gAMGUCMQDwm3Hce0BWn/9hqTq6bfEZEERPiPmA
/6WGbxOJyBl1OZ4OtuiFu58GDQcxEbqAVysCMAi54FIpj4kUhCjHJ4AdmHPqlyxr
MeOEt6xIsdZU1Uk1VMpmJ4969uKxHjiriqXzhg==
-----END CERTIFICATE-----
```

SubCA Certificate: Cisco ECC SUDI CA

```
-----BEGIN CERTIFICATE-----
MIIDETCCApagAwIBAgIBBTAKBggqhkjOPQQDAzAsMQ4wDAYDVQQKEwVDaXNjbzEa
MBgGA1UEAxMRQ2lzY28gRUNDIFJvb3QgQ0EwIBcNMTMwNDA0MDgyNjEzWhgPMjA5
OTA5MDcxNjI0MDZaMCsxDjAMBgNVBAoTBUNpc2NvMRkwFwYDVQQDExBBQ1QyIEVD
QyBTVURJIENBMHYwEAYHKoZIzj0CAQYFK4EEACIDYgAEjGHc+nnSMQr4zuARzDxH
F6TDi97f7eGqSdDK5misVpoBP0Nwwcmazs+GTiT+Us52/BKuo4JlaZ74cnlzzBIn
kPwtuag2bXSCeYVJhcB3sVuVvYiSKVEi1OIgx85dtXdwo4IBiTCCAYUwDgYDVR0P
AQH/BAQDAgEGMBIGA1UdEwEB/wQIMAYBAf8CAQAwfQYIKwYBBQUHAQEEcTBvMD8G
CCsGAQUFBzAChjNodHRwOi8vd3d3LmNpc2NvLmNvbS9zZWN1cml0eS9wa2kvY2Vy
dHMvZWNjcm9vdC5jZXIwLAYIKwYBBQUHMAGGIGh0dHA6Ly9wa2ljdnMuY2lzY28u
Y29tL3BraS9vY3NwMB8GA1UdIwQYMBaAFKRFti+jMbF2FbAKGDPK9q1PPSgEMFwG
A1UdIARVMFMwUQYKKwYBBAEJFQETADBDMEEGCCsGAQUFBwIBFjVodHRwOi8vd3d3
LmNpc2NvLmNvbS9zZWN1cml0eS9wa2kvcG9saWNpZXMvaW5kZXguaHRtbDBCBgNV
HR8EOzA5MDegNaAzhjFodHRwOi8vd3d3LmNpc2NvLmNvbS9zZWN1cml0eS9wa2kv
Y3JsL2VjY3Jvb3QuY3JsMB0GA1UdDgQWBBSWhzrYiYGRQRUzv+A0jyCPwrvDljAK
BggqhkjOPQQDAwNpADBmAjEAzye9v1h5m/1bAUSU0MQwShDDCRJHXsc+1TtbqOJR
1cZMmhsIzNNy/AskkBoIgNNtAjEAk3DAKxkBx3DYyZS4Uz3B50MpbrZwpxPTUCqW
m7qExCM7m1FjIg98ltfdltPHgqP/
-----END CERTIFICATE-----
```

Use these certificates as the trust anchors when verifying the IAK certificate signature for devices using the ECC P384 algorithm.

- Confirm that the device identity fields in the IAK and IDevID certificates meet the expected criteria.

- Confirm that the device identity fields in the IAK and IDevID certificates meet the expected criteria.

Step 4 Request and install owner certificates:

- Request the router owner CA to issue the Owner IAK (oIAK) and Owner IDevID (oIDevID) certificates based on the public keys.
- Use the `RotateOIakCert` API to install the oIAK and oIDevID certificates on the control card.

Step 5 Verify and Store certificates:

- Verify that the public keys in the oIAK and oIDevID certificates match with respective IAK and SUDI public key.
- Store the oIAK and oIDevID certificates in non-volatile memory for presentation during the TPM attestation (`attestz`) workflow.

Step 6 Update SSL profile: Update the SSL profile to use the trust bundle and rotate the certificates to the Owner IDevID certificate.

Step 7 Enroll secondary control card: Repeat the enrollment workflow for the secondary control card, if present.

Step 8 Obtain the attestation result from the device and verify that the Platform Configuration Register (PCR) values match the expected known-good values, confirming that the device has securely booted with the approved firmware and software artifacts.

The TPM 2.0 is successfully enrolled and ready for attestation workflows using the installed certificates.

Example

For example, after enrollment, the device presents the oIAK and oIDevID certificates during attestation to prove its identity and integrity.

What to do next

After completing this task:

- Monitor the attestation logs to ensure the enrolled certificates are used and validated successfully.

TPM 2.0 attestation

The TPM 2.0 attestation workflow ensures the integrity and identity of network devices by verifying their configurations and credentials.

This process involves interaction with gRPC TPM 2.0 attestation endpoints and requires the device to be booted with the correct OS image and configurations.

Before you begin

- Device must be booted with the correct OS image.
- Correct configurations and credentials must be applied.
- Primary/active control card is responsible for all RPCs directed to the secondary/standby control card.

Procedure

Step 1 Serve gRPC TPM 2.0 Attestation Endpoints: Ensure the device serves gRPC TPM 2.0 attestation endpoints on port 9339, the same port as gNOI/gNSI/gNMI.

The device must be booted with the correct OS image and configurations.

Step 2 Authenticate Standby Control Card: Perform an authentication handshake between the active and standby control cards using the IDevID key pair/cert.

The active control card is responsible for this handshake as the router owner cannot directly TLS authenticate the standby card.

Step 3 Secure Initial Attestation RPCs: Use the active control card's IDevID private key and oIDevID cert to secure TLS for the initial attestation RPCs.

Step 4 Call AttestZ Service: AttestZ service calls the device's Attest endpoint for a given control card (and a random nonce) to get back:

- An oIAK cert signed by the router owner’s CA.
- Final observed PCR hashes/values.
- PCR Quote structure and signature over it signed by IAK private key.
- (Optional) oIDevID cert of the standby control card.

Step 5 Verify Certificates and Signatures:

- AttestZ service uses the trust bundle/anchor from the router owner CA to verify the oIAK cert and its validity/revocation status.
- Ensure that the control card serial number in the oIAK cert and oIDevID cert is the same.

Step 6 Compare PCR Values: The AttestZ service compares the PCR values against the known PCR values provided by the OEM vendor specific to a release.

Step 7 Compare PCR Values and Record Attestation Status: AttestZ service fetches expected final PCR values from its database and compares them to the observed ones reported by the device.

AttestZ service records a successful attestation status for the given control card and repeats the workflow for the secondary/standby control card if one is available.

The TPM 2.0 attestation process completes successfully, verifying the device's integrity and recording the attestation status for each control card.

Example

For example, if the standby control card is present, the AttestZ service will repeat the attestation steps to verify its identity and configuration using the same process as for the active control card.

What to do next

After completing this task:

- Monitor the attestation logs and ensure that both control cards have passed the verification process.