***Intel OpenBMC SPDM Design Document***

Revision: 0.4

**Document Change History**

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Version** | **Author** | **Description** |
| 03/04/2022 | 0.1 |  | First Draft |
| 03/16 | 0.2 |  |  |
| 05/03 | 0.3 |  | Remove SPDM requester |
| 05/10 | 0.4 |  | Add SPDM requester |
|  |  |  |  |
|  |  |  |  |

Contents

[1. Purpose 5](#_Toc103082738)

[2. Introduction 5](#_Toc103082739)

[3. Requirements 6](#_Toc103082740)

[4. Architectural Design 9](#_Toc103082741)

[3. SPDMDesign 10](#_Toc103082742)

[3.1 SPDMAPPLib 10](#_Toc103082743)

[Public Member Functions 12](#_Toc103082744)

[Constructor & Destructor 13](#_Toc103082745)

[Member Function 14](#_Toc103082746)

[3.2 SPDM Configuration file 18](#_Toc103082747)

# Purpose

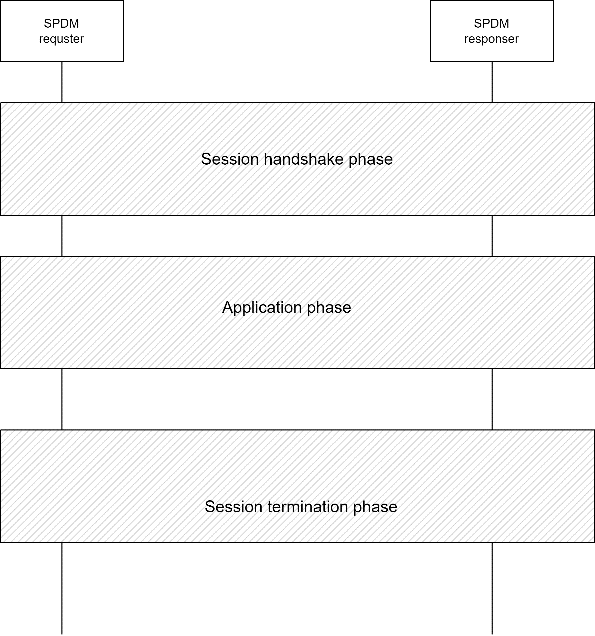
The purpose of the document is to define the required working items/jobs of SPDM module in Intel OpenBMC. The function outside the BMC FW is not in the scope of this document.

# 2. Introduction

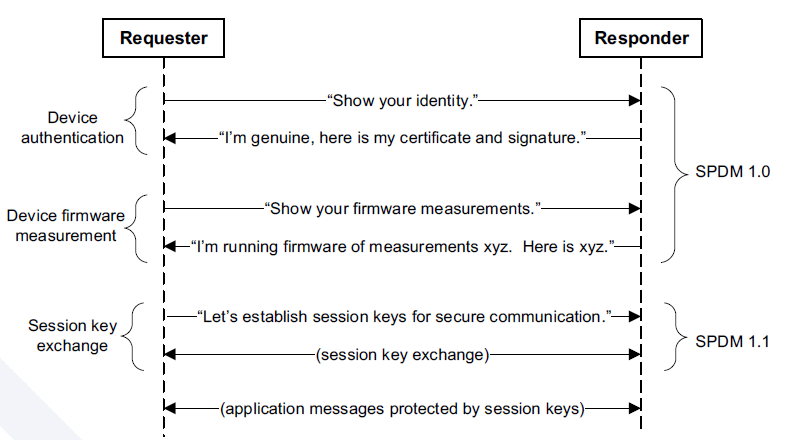
The Security Protocol and Data Model (SPDM) defines messages, data objects, and sequences for performing message exchanges between devices over a variety of transport and physical media. The description of message exchanges includes authentication of hardware identities, measurement for firmware identities and session

Key exchange protocols to enable confidentiality and integrity protected data communication.

There are three phases in a session, as Session phases shows: the handshake, the application, and termination.



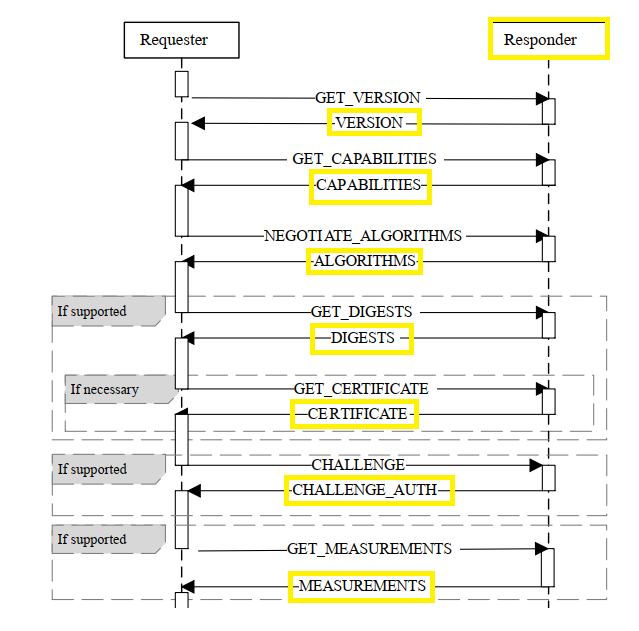
In session handshake phase:



# Requirements

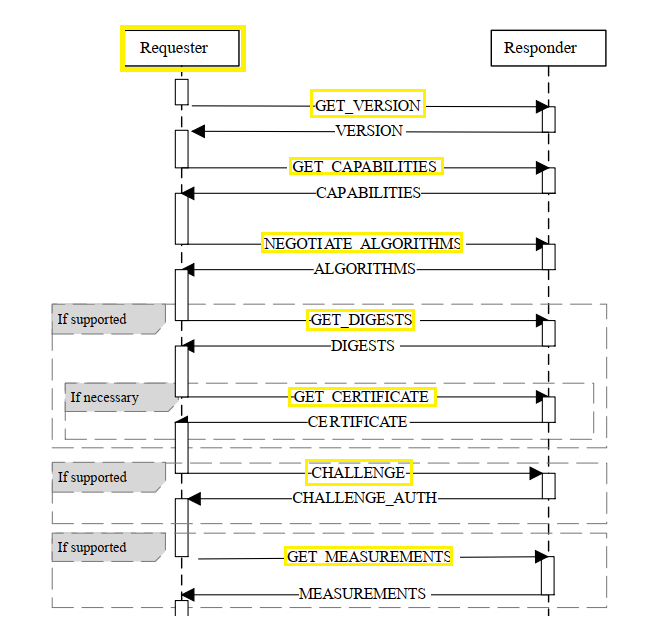
Intel OpenBMC SPDM module design and implementation requirements are as below. OpenBMC SPDM module supports both SPDM requester and responder roles in parallel. It also supports SPDM requester and responder capabilities as configurable, such that the roles of BMC as SPDM device and other SPDM settings can be statically configured.

1. BMC as SPDM Responder
   1. Intel OpenBMCSPDM module supports responder provides hook functions to all the below SPDM commands as a responder.



* 1. OpenBMC SPDM responder module integrates with existing OpenBMC’s firmware utilities in order to select firmware regions and to calculate firmware measurements.

1. BMC as SPDM Requester
   1. SPDM Requester module to support below commands



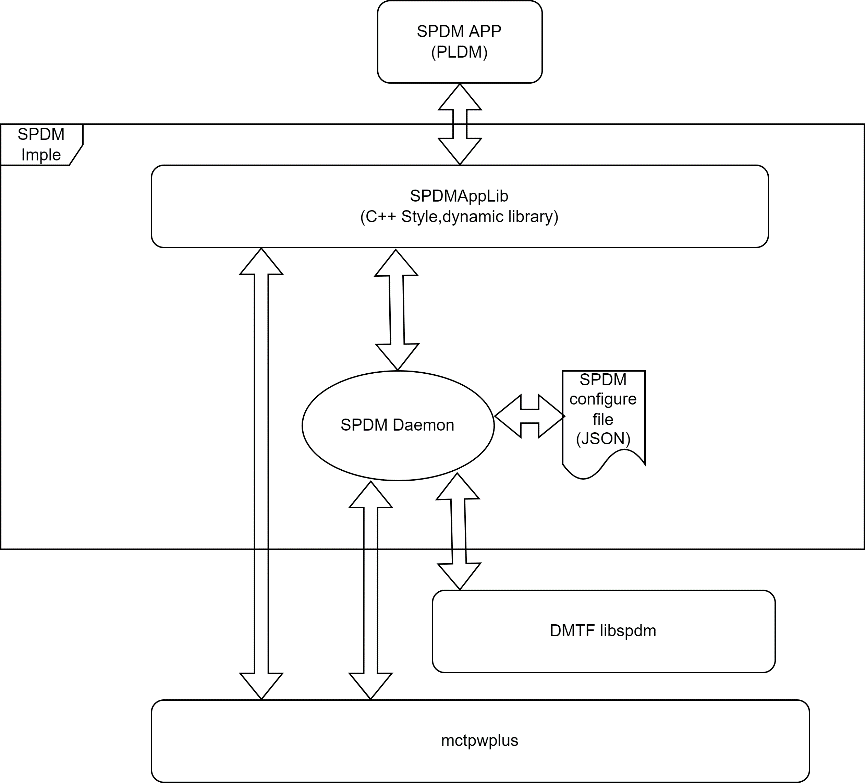
* 1. SPDM Requester module supports verifier library which maintains reference measurements in order to support attestations.
  2. SPDM Requester module provides requester commands proxy support for external users to send commands directly to SPDM devices over Redfish.

# Architectural Design

Intel OpenBMC SPDM module includes:

* SPDM Daemon
  + The Daemon that makes BMC as either SPDM Requester or Responder or both.
  + Daemon that triggers individual services to handle requester or responder role.
* SPDM Application Library (SPDMAppLib)
  + SPDMAppLib handles both Requester and Responder framework.
  + SPDMAppLib integrates with both MCTPW+ Library and with libSPDM.
  + SPDMAppLib is dynamic C++ library that registers with C-Style transport layer with libSPDM.
  + SPDLAppLib invokes C++ style MCTPWPlus APIs. Also it invokes C libSPDM library for SPDM functions.





# 3. SPDMDesign

## 3.1 SPDMAPPLib

#include <spdmapplib.hpp>

1

17 #pragma once

18 #include "mctp\_wrapper.hpp"

19

20 #include <stdio.h>

21

22 #include <sdbusplus/asio/connection.hpp>

23

24 #include <chrono>

25 #include <cstdint>

26 #include <functional>

27 #include <optional>

28 #include <string>

29 #include <vector>

30

31 class SPDMAPPLib

32 {

33 using ReceiveCallback =

34 std::function<void(boost::system::error\_code, mctpw::ByteArray&)>;

35 using SendCallback = std::function<void(boost::system::error\_code, int)>;

36

37 public:

48 SPDMAPPLib(boost::asio::io\_context& ioContext,

49 const mctpw::ReconfigurationCallback& networkChangeCb,

50 const mctpw::ReceiveMessageCallback& rxCb);

61 SPDMAPPLib(std::shared\_ptr<sdbusplus::asio::connection> conn,

62 const mctpw::ReconfigurationCallback& networkChangeCb,

63 const mctpw::ReceiveMessageCallback& rxCb);

69 std::string getEndpointStatus(const mctpw::eid\_t eid);

75 std::vector<mctpw::eid\_t> getEndpointMap();

88 void sendAsync(const SendCallback& callback, const mctpw::eid\_t dstEId,

89 const uint8\_t msgTag, const bool tagOwner,

90 const mctpw::ByteArray& request);

103 std::pair<boost::system::error\_code, int>

104 sendYield(boost::asio::yield\_context& yield, const mctpw::eid\_t dstEId,

105 const uint8\_t msgTag, const bool tagOwner,

106 const mctpw::ByteArray& request);

116 void sendReceiveAsync(ReceiveCallback receiveCb, mctpw::eid\_t dstEId,

117 const mctpw::ByteArray& request,

118 std::chrono::milliseconds timeout);

129 std::pair<boost::system::error\_code, mctpw::ByteArray>

130 sendReceiveYield(boost::asio::yield\_context yield, mctpw::eid\_t dstEId,

131 const mctpw::ByteArray& request,

132 std::chrono::milliseconds timeout);

133

143 int doAuthentication(eid\_tdstEId);

144

152 int doMeasurement(eid\_tdstEId);

153

154 private:

155 std::unique\_ptr<mctpw::MCTPWrapper> mctpWrapper;

156 };

### Public Member Functions

* **SPDMAPPLib** (boost::asio::io\_context &ioContext, const mctpw::ReconfigurationCallback &networkChangeCb, const mctpw::ReceiveMessageCallback &rxCb)

*Construct a new* ***SPDMAPPLib*** *object.*

* **SPDMAPPLib** (std::shared\_ptr< sdbusplus::asio::connection > conn, const mctpw::ReconfigurationCallback &networkChangeCb, const mctpw::ReceiveMessageCallback &rxCb)

*Construct a new* ***SPDMAPPLib*** *object.*

* std::string **getEndpointStatus** (const mctpw::eid\_t eid)

*Get status EndpointMap.*

* std::vector< mctpw::eid\_t > **getEndpointMap** ()

*Get a reference to internaly maintained EndpointMap.*

* void **sendAsync** (const SendCallback &callback, const mctpw::eid\_t dstEId, const uint8\_t msgTag, const bool tagOwner, const mctpw::ByteArray &request)

*Send MCTP request to dstEId and receive status of send operation in callback through SPDM.*

* std::pair< boost::system::error\_code, int > **sendYield** (boost::asio::yield\_context &yield, const mctpw::eid\_t dstEId, const uint8\_t msgTag, const bool tagOwner, const mctpw::ByteArray &request)

*Send MCTP request to dstEId and receive status of send operation through SPDM.*

* void **sendReceiveAsync** (ReceiveCallback receiveCb, mctpw::eid\_t dstEId, const mctpw::ByteArray &request, std::chrono::milliseconds timeout)

*Send request to dstEId and receive response asynchronously in receiveCb through SPDM.*

* std::pair< boost::system::error\_code, mctpw::ByteArray > **sendReceiveYield** (boost::asio::yield\_context yield, mctpw::eid\_t dstEId, const mctpw::ByteArray &request, std::chrono::milliseconds timeout)

*Send request to dstEId and receive response using yield\_context throught SPDM.*

* int **doAuthentication** (eid\_tdstEId)

*This function sends GET\_DIGEST, GET\_CERTIFICATE, CHALLENGE to authenticate the device.*

* int **doMeasurement** (eid\_tdstEId)

*This function executes SPDM measurement and extend to TPM.*

### Constructor & Destructor

**SPDMAPPLib::SPDMAPPLib (boost::asio::io\_context & *ioContext*, const mctpw::ReconfigurationCallback & *networkChangeCb*, const mctpw::ReceiveMessageCallback & *rxCb*)**

Construct a new **SPDMAPPLib** object.

* Parameters

|  |  |
| --- | --- |
| *ioContext* | boost io\_context object. Usable if invoker is an sdbus unaware app. |
| *networkChangeCb* | Callback to be executed when a network change occurs in the system. For example a new device is inserted or removed etc |
| *rxCb* | Callback to be executed when new MCTP message is received. |

**SPDMAPPLib::SPDMAPPLib (std::shared\_ptr< sdbusplus::asio::connection > *conn*, const mctpw::ReconfigurationCallback & *networkChangeCb*, const mctpw::ReceiveMessageCallback & *rxCb*)**

Construct a new **SPDMAPPLib** object.

* Parameters

|  |  |
| --- | --- |
| *ioContext* | boost io\_context object. Usable if invoker is an sdbus unaware app. |
| *networkChangeCb* | Callback to be executed when a network change occurs in the system. For example a new device is inserted or removed etc |
| *rxCb* | Callback to be executed when new MCTP message is received. |

### Member Function

**std::vector< mctpw::eid\_t > SPDMAPPLib::getEndpointMap ()**

Get a reference to internaly maintained EndpointMap.

* Returns

std::vector<mctpw::eid\_t>

**std::string SPDMAPPLib::getEndpointStatus (const mctpw::eid\_t *eid*)**

Get status EndpointMap.

* Returns

std::string status

**void SPDMAPPLib::sendAsync (const SendCallback & *callback*, const mctpw::eid\_t *dstEId*, const uint8\_t *msgTag*, const bool *tagOwner*, const mctpw::ByteArray & *request*)**

Send MCTP request to dstEId and receive status of send operation in callback through SPDM.

* Parameters

|  |  |
| --- | --- |
| *callback* | Callback that will be invoked with status of send operation |
| *dstEId* | Destination MCTP Endpoint ID |
| *msgTag* | MCTP message tag value |
| *tagOwner* | MCTP tag owner bit. Identifies whether the message tag was originated by the endpoint that is the source of the message |
| *request* | MCTP request byte array |

**void SPDMAPPLib::sendReceiveAsync (ReceiveCallback *receiveCb*, mctpw::eid\_t *dstEId*, const mctpw::ByteArray & *request*, std::chrono::milliseconds *timeout*)**

Send request to dstEId and receive response asynchronously in receiveCb through SPDM.

* Parameters

|  |  |
| --- | --- |
| *receiveCb* | Callback to be executed when response is ready |
| *dstEId* | Destination MCTP Endpoint ID |
| *request* | MCTP request byte array |
| *timeout* | MCTP receive timeout |

* Returns

std::pair< boost::system::error\_code, mctpw::ByteArray >

**SPDMAPPLib::sendReceiveYield (boost::asio::yield\_context *yield*, mctpw::eid\_t *dstEId*, const mctpw::ByteArray & *request*, std::chrono::milliseconds *timeout*)**

Send request to dstEId and receive response using yield\_context throught SPDM.

* Parameters

|  |  |
| --- | --- |
| *yield* | Boost yield\_context to use on dbus call |
| *dstEId* | Destination MCTP Endpoint ID |
| *request* | MCTP request byte array |
| *timeout* | MCTP receive timeout |

* Returns

std::pair<boost::system::error\_code, ByteArray> Pair of boost error code and response byte array

**std::pair< boost::system::error\_code, int > SPDMAPPLib::sendYield (boost::asio::yield\_context & *yield*, const mctpw::eid\_t *dstEId*, const uint8\_t *msgTag*, const bool *tagOwner*, const mctpw::ByteArray & *request*)**

Send MCTP request to dstEId and receive status of send operation through SPDM.

* Parameters

|  |  |
| --- | --- |
| *yield* | boost yiled\_context object to yield on dbus calls |
| *dstEId* | Destination MCTP Endpoint ID |
| *msgTag* | MCTP message tag value |

**int SPDMAPPLib::doAuthentication (eid\_tdstEId )**

This function sends GET\_DIGEST, GET\_CERTIFICATE, CHALLENGE to authenticate the device.

* Parameters

|  |  |
| --- | --- |
| *dstEId* | Destination MCTP Endpoint ID |

* Returns

RETURN\_SUCCESS : The authentication is got successfully. RETURN\_DEVICE\_ERROR: A device error occurs when communicates with the device. RETURN\_SECURITY\_VIOLATION:Any verification fails.

**int SPDMAPPLib::doMeasurement (eid\_tdstEId )**

This function executes SPDM measurement and extend to TPM.

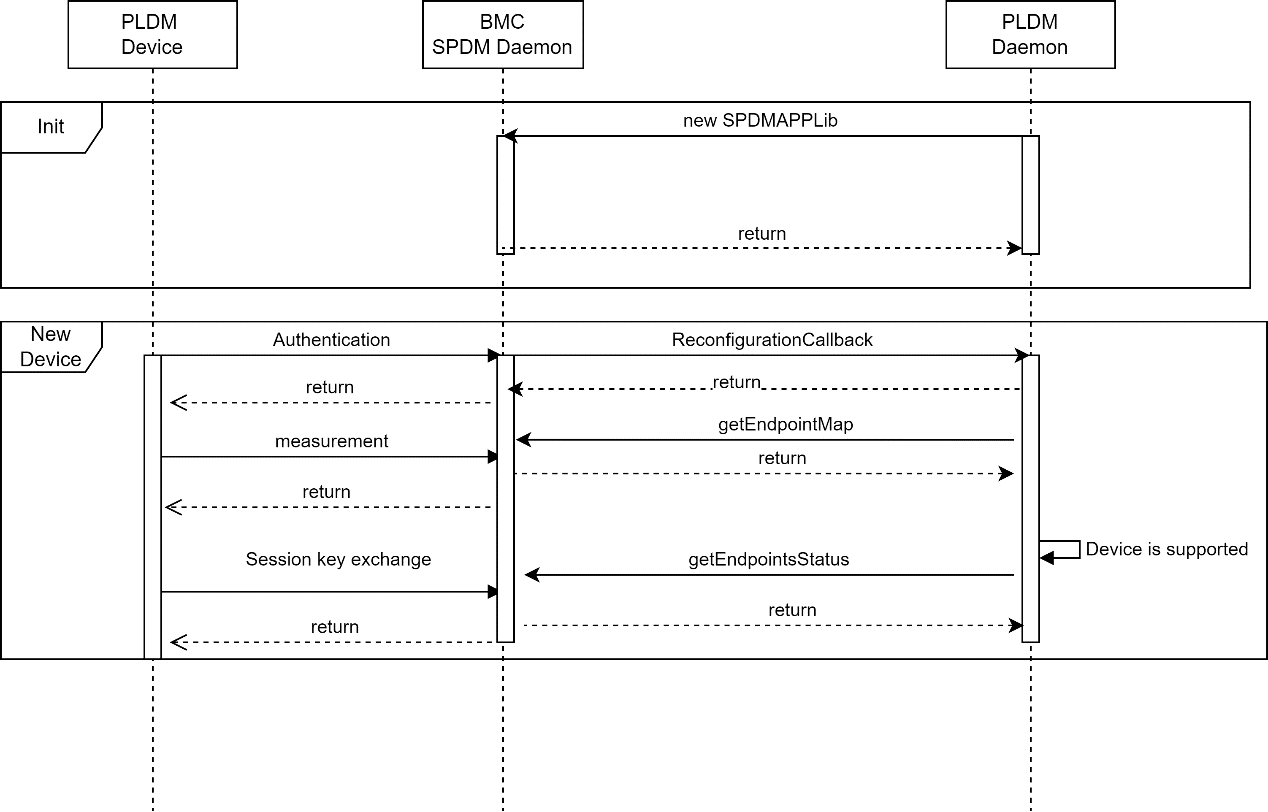
* Parameters

|  |  |
| --- | --- |
| *dstEId* | Destination MCTP Endpoint ID |

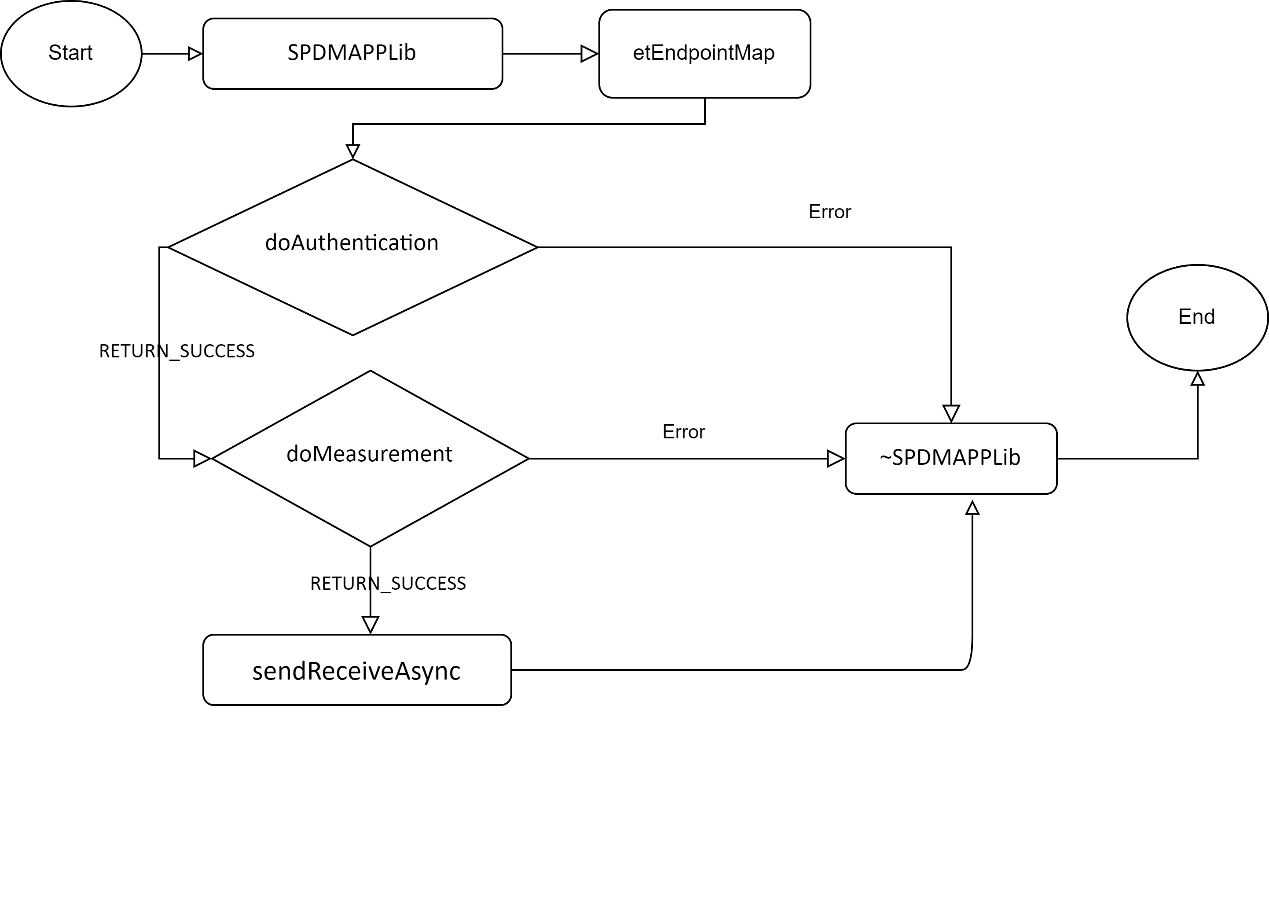
* Returns

RETURN\_SUCCESS : The authentication is got successfully. RETURN\_DEVICE\_ERROR: A device error occurs when communicates with the device.

APP of SPDM responder:



APP of SPDM requester:



## 3.2 SPDM Configuration file

Add to entity-manager:

{

"Role": "responder",

"Version": "1.0",

"CertPath": "/usr/bin",

"Capability": [

"CACHE",

"CERT",

"CHAL",

"MEAS\_SIG",

"MEAS\_FRESH",

"ENCRYPT",

"MAC",

"MUT\_AUTH",

"KEY\_EX",

"PSK\_WITH\_CONTEXT",

"ENCAP","HBEAT",

"KEY\_UPD",

"HANDSHAKE\_IN\_CLEAR"

],

"Hash": [

"SHA\_256",

"SHA\_384",

"SHA\_512",

"SHA3\_256",

"SHA3\_384",

"SHA3\_512",

"SM3\_256"

],

"MeasHash": [

"SHA\_256",

"SHA\_384",

"SHA\_512",

"SHA3\_256",

"SHA3\_384",

"SHA3\_512",

"SM3\_256"

],

"Asym": [

"ECDSA\_P384",

"ECDSA\_P256"

],

"Dhe": [

"SECP\_384\_R1",

"SECP\_256\_R1",

"FFDHE\_3072","FFDHE\_2048"

],

"Aead": [

"AES\_256\_GCM",

"CHACHA20\_POLY1305"

],

"BasicMutAuth": "BASIC",

"MutAuth": "W\_ENCAP",

"Index": 4,

"Type": "SPDMConfiguration",

"Name": "SPDM responder"

},

{

"Role": "Requester",

"Version": "1.0",

"CertPath": "/usr/bin",

"Capability": [

"CACHE",

"CERT",

"CHAL",

"MEAS\_SIG",

"MEAS\_FRESH",

"ENCRYPT",

"MAC",

"MUT\_AUTH",

"KEY\_EX",

"PSK\_WITH\_CONTEXT",

"ENCAP","HBEAT",

"KEY\_UPD",

"HANDSHAKE\_IN\_CLEAR"

],

"Hash": [

"SHA\_256",

"SHA\_384",

"SHA\_512",

"SHA3\_256",

"SHA3\_384",

"SHA3\_512",

"SM3\_256"

],

"MeasHash": [

"SHA\_256",

"SHA\_384",

"SHA\_512",

"SHA3\_256",

"SHA3\_384",

"SHA3\_512",

"SM3\_256"

],

"Asym": [

"ECDSA\_P384",

"ECDSA\_P256"

],

"Dhe": [

"SECP\_384\_R1",

"SECP\_256\_R1",

"FFDHE\_3072","FFDHE\_2048"

],

"Aead": [

"AES\_256\_GCM",

"CHACHA20\_POLY1305"

],

"BasicMutAuth": "BASIC",

"MutAuth": "W\_ENCAP",

"Index": 4,

"Type": "SPDMConfiguration",

"Name": "SPDM requester"

}

* [role] is SPDM role.
  + Responder/Requester.
* [CertPath] is path of SPDM certification files
* [version] is SPDM version.
  + 1.0|1.1|1.2
* [capability] is capability flags.
  + CACHE|CERT|CHAL|MEAS\_NO\_SIG|MEAS\_SIG|MEAS\_FRESH|ENCRYPT|MAC|MUT\_AUTH|KEY\_EX|PSK|PSK\_WITH\_CONTEXT|ENCAP|HBEAT|KEY\_UPD|HANDSHAKE\_IN\_CLEAR|PUB\_KEY\_ID|CHUNK|ALIAS\_CERT
* [hash] is hash algorithm.
  + SHA\_256|SHA\_384|SHA\_512|SHA3\_256|SHA3\_384|SHA3\_512|SM3\_256
* [--meas\_hash] is measurement hash algorithm
  + RAW\_BIT|SHA\_256|SHA\_384|SHA\_512|SHA3\_256|SHA3\_384|SHA3\_512|SM3\_256
* [asym] is asym algorithm
  + RSASSA\_2048|RSASSA\_3072|RSASSA\_4096|RSAPSS\_2048|RSAPSS\_3072|RSAPSS\_4096|ECDSA\_P256|ECDSA\_P384|ECDSA\_P521|SM2\_P256|EDDSA\_25519|EDDSA\_448
* [dhe] is DHE algorithm
  + FFDHE\_2048|FFDHE\_3072|FFDHE\_4096|SECP\_256\_R1|SECP\_384\_R1|SECP\_521\_R1|SM2\_P256
* [aead] is AEAD algorithm
  + AES\_128\_GCM|AES\_256\_GCM|CHACHA20\_POLY1305|SM4\_128\_GCM
* [basic\_mut\_auth] is the basic mutual authentication policy
  + NO|BASIC
* [mut\_auth] is the mutual authentication policy
  + NO|WO\_ENCAP|W\_ENCAP|DIGESTS