Document Title	System Tests of Adaptive Platform
Document Owner	AUTOSAR
Document Responsibility	AUTOSAR
Document Identification No	890

Document Status	published
Part of AUTOSAR Standard	Adaptive Platform
Part of Standard Release	R21-11

Document Change History

Date	Release	Changed by	Description
2021-11-25	R21-11	AUTOSAR Release Management	• Added test cases for LT, E2E and CRYPTO • Added new sections for Health Management and State Management • Removed REST test cases
2020-11-30	R20-11	AUTOSAR Release Management	• Added test cases for CM, REST, EMO, DIAG, PER, IAM, UCM and CRYPTO • Added cross reference links to corresponding Test Configuration in the test cases • Updated limitations for CRYPTO • Removed acronyms which are already part of AUTOSAR glossary
2019-11-28	R19-11	AUTOSAR Release Management	• Changed format for Actors (App, Events, Services etc.) • Added new sections and test cases for Security Management, Network Management and Cryptography • Added more test cases for CM, EMO, TS, and E2E • Changed Document Status from Final to published

2019-03-29	19-03	AUTOSAR Release Management	• Changed format for RS traceability items • Added new section and test cases for Time Synchronization • Added more test cases for CM, EMO, and DIAG
2018-10-31	18-10	AUTOSAR Release Management	• Added RS traceability for test cases • Added ISO 9646 framework and mapping on system test architecture • Added more test cases for CM, REST, EMO, and UCM
2018-03-31	18-03	AUTOSAR Release Management	• Test case for RESTful communication is added • Test case for Security is added • Test case for Update and Configuration Management is added • Test case for E2E is added
2017-10-27	17-10	AUTOSAR Release Management	• Initial release

Table of Contents

1 Acronyms and abbreviations

The glossary below includes terms, acronyms and abbreviations relevant to

System Test Specification that are not included in the AUTOSAR Glossary (see References).

Abbreviation / Acronym:	**Description:**
IUT	Implementation Under Test
NRC	Negative Response Code
RS	Requirement Specification
SM	State Manager
ST	System Test

2 Scope of Document

The system test cases are used to validate RS items in order to confirm whether requirements of functional cluster are satisfied by the AUTOSAR Adaptive Platform Demonstrator. Each test case is applicable with the coupled specification release.

In this release, Requirement Specifications of CM (someip), EMO, DIA, LT, PER, IAM, UCM, E2E, TS, SEC, NM and CRYPTO are in the scope of this document.

2.1 Supported hardware

For the current release, **Raspberry Pi 3 Model B** and **Raspberry Pi 4** shall be the supported hardware for test configurations.

2.2 Overview of test architecture

In this section, System Test architecture is described according to ISO 9646 test architecture manner. In System Test, FC tester is called as LT (Lower Tester) which stimulate and observe IUT (Implementation Under Test) behavior. AP instances is called as IUT (Implementation Under Test) which is the test target. Applications is called as UT (Upper Tester) which is stimulated by LT (Lower Tester) and take an action to request test step (e.g. sending message) to IUT.

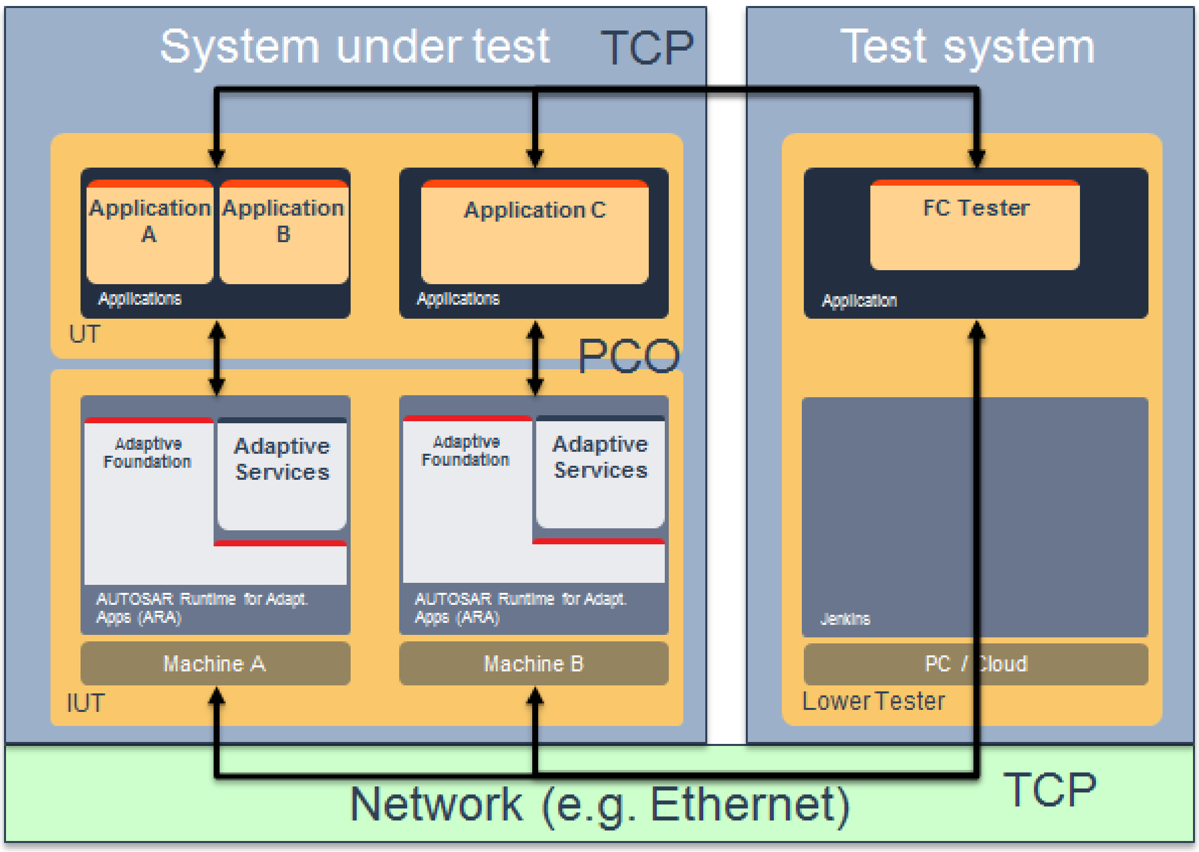

Figure 2.1: System Test architecture

The following picture describing that mapping to System Test implementation. In ST demonstrator, TCP (Test Coordination Procedures) is realized by stimulating application via Diagnostics routine service. PCO (Point of Control and Observation) is realized by requesting action via ARA::API, and receive/ transmit Ethernet message so that IUT could react. Application send message after certain step is passed so that test system could observe what happens on System under test.

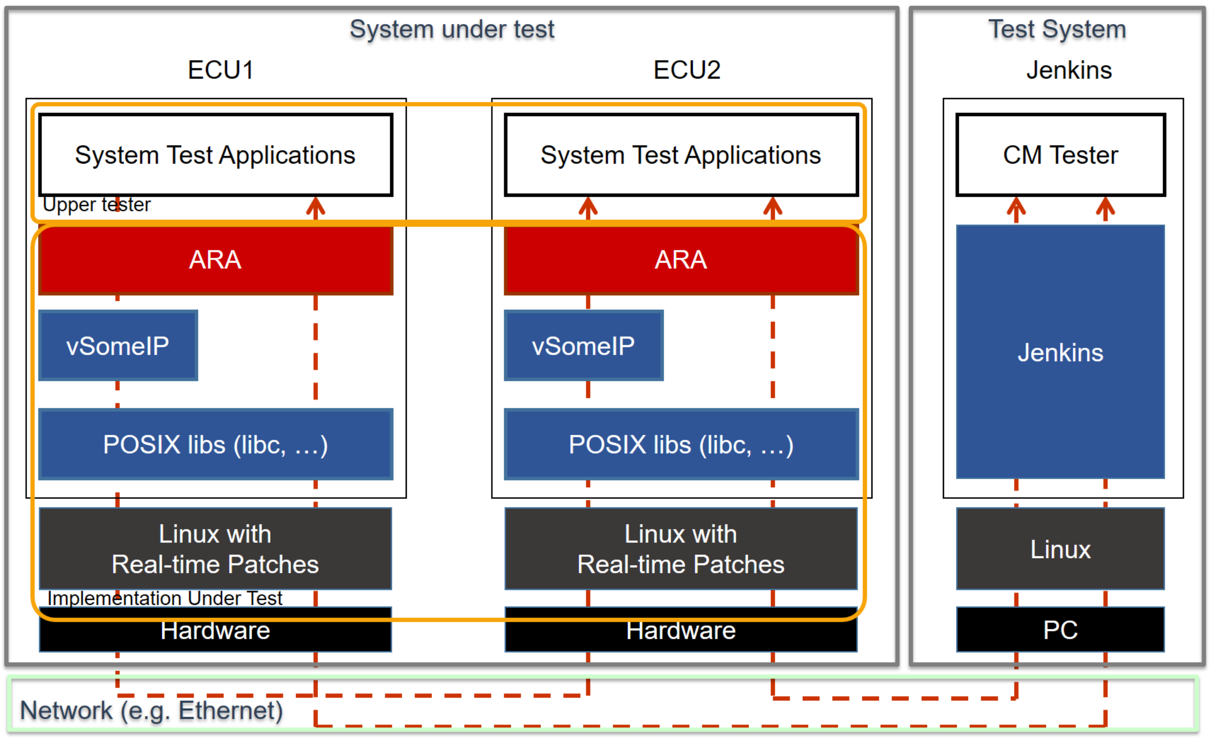

Figure 2.2: Map to System Test implementation

3 Limitations

There are several limitations in this document.

- Test cases may not cover whole RS as specified against test cases
- Test Setup and configurations are for reference purpose only and may cover broader scope than represented by test cases in corresponding sections
- Test cases may not be fully covered by corresponding system test implementations
- System test cases are just examples, since there could be many ways to define and implement use case scenarios
- DIAG traceability is obsolete as SRS is changed to RS
- LT does not have any RS traceability. Traceability will be added in next release
- In the E2E test case, the common parts of the E2E profiles are checked
- Time Base (TB) of Time Synchronization has five TB types. (Synchronized Master TB, Offset Master TB, Synchronized Slave TB, Offset Slave TB, Pure Local TB.) RS_TimeSynchronization describes multiple TB types as scope, but system test cases may not cover whole TB types.
- In Cryptography test cases [STS_CRYPTO_00002] Encrypting and decrypting data using an algorithm for asymmetric encryption/decryption primitives and [STS_CRYPTO_00004] Generation and verification of digital signature, both public and private keys are used by the test application to simplify the test case (i.e. not corresponding to practical use of asymmetric keys)
- In Cryptography test case [STS_CRYPTO_00006] Generation of random number, only deterministic random number generation is tested; true random number generation is not in the scope of the system test.
- Even if the behaviour is different, same application and/or service numbers are used across different test cases

4 Test configuration and test steps for Communication Management

4.1 Test System

4.1.1 Test configurations Communication Management

Configuration ID	STC_CM_00001
Description	Standard Jenkins server for Communication Management test
ECU 1	Hardware, 192.168.100.5
ECU 2	Hardware, 192.168.100.2
Jenkins	Jenkins Server, 192.168.100.10

Configuration ID	STC_CM_00002
Description	Scenario 2 Variant 2 - Reference Deployment
ECU 1	Hardware, 192.168.100.5
ECU 2	Hardware, 192.168.100.2
Jenkins	Jenkins Server, 192.168.100.10

The Jenkins Server, running the job with the Communication Management test ([CM Tester]) is connected via Ethernet to [ECU1] hosting the System Test Application [CMApp01] (as well as [CMApp04] on the alternative configuration) and [ECU2] hosting the System Test Applications [CMApp02], [CMApp03], [CMApp04] and [CMApp05].

The [CM Tester] is supposed to collect the results.

The communication between [CM Tester] and the applications on the ECU may take place over the Diagnostics functional cluster in form of diagnostic messages.

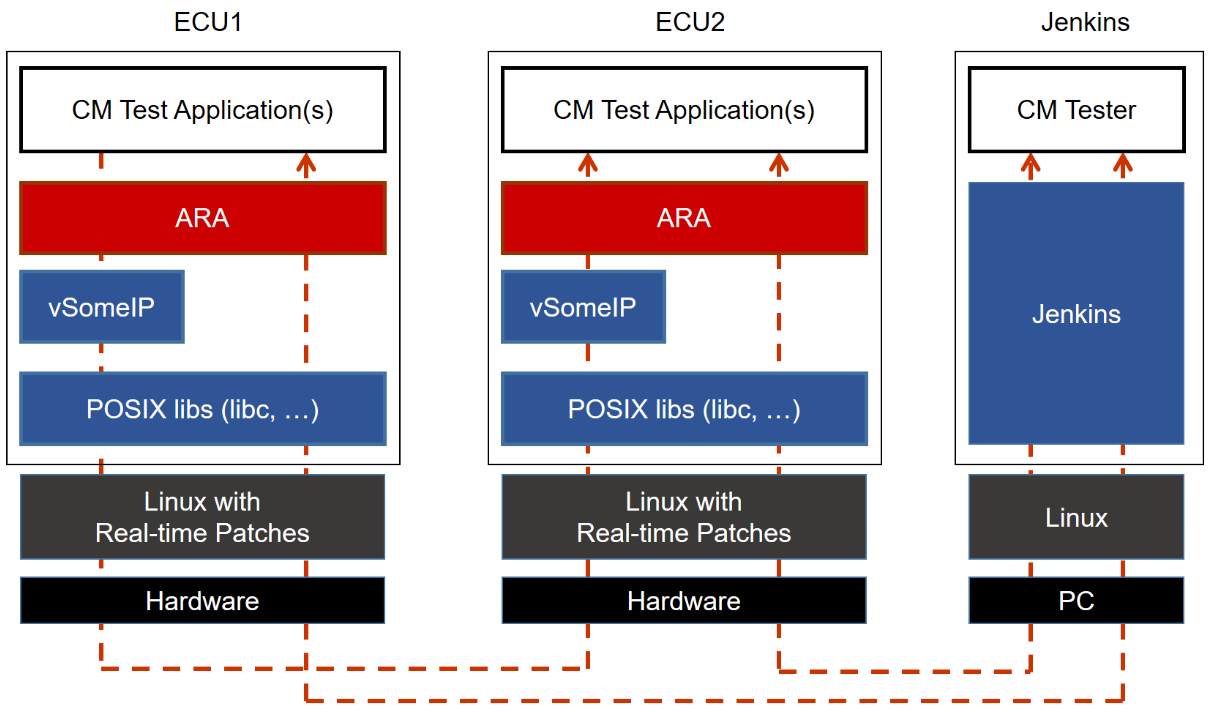

Figure 4.1: Illustration of test setup for Communication Management

4.1.2 Test configurations SignalToService

Configuration ID	STC_S2S_00001
Description	Test configuration for SignalToService Translation testcases.
ECU 1	Hardware, 192.168.100.5
ECU 2	Classic platform ECU, 192.168.100.3
Jenkins	Jenkins Server, 192.168.100.10

The Jenkins Server, running the [CM Tester] is connected to ECU1 - Adaptive Platform and ECU2 - Classic Platform.

The [CM Tester] is supposed to collect the results.

4.2 Test cases

4.2.1 [STS_CM_00001] Local and remote service discovery.

Test Objective	To verify that the applications are able to offer, request and stop services and that service discovery works, establishing the correct communication paths.		
ID	STS_CM_00001	**State**	Draft
Affected Functional Cluster	Communication Management		
Trace to RS Criteria	[RS_CM_00101], [RS_CM_00102], [RS_CM_00105], [RS_CM_00107], [RS_CM_00211]		
Reference to Test Environment	STC_CM_00001 in Test configurations Communication Management		
Configuration Parameters	- The existing communication services comprise the following (service names are arbitrary): - [CMService01]: Offered by [CMApp02], requested by [CMApp01]. - [CMService02]: Offered by [CMApp02], requested by [CMApp03]. - [CMService03]: Offered by [CMApp01], requested by [CMApp02]. - [CMService04]: Not available, requested by [CMApp03]. - [CMService01], [CMService02], [CMService03] and [CMService04] are attributes of Methods, Events and Fields.		
Summary	First, the [CMApp02] and [CMApp03] applications on [ECU2] are started when Machine State for [ECU2] is changed to Driving. The [CMApp02] offers the services [CMService01] and [CMService02] and requests the service [CMService03]. [CMApp03] requests the service [CMService02]. The [CM Tester] trigger application [CMApp02] to Stop Offering service [CMService02]. Then [CMApp02] again offer service [CMService02] and initial reconnection is established between [CMApp02] and [CMApp03]. Then the [CMApp01] application on [ECU1] is started when Machine State for [ECU1] is changed to Driving. The [CMApp01] offers the service [CMService03] and requests the service [CMService01]. [CMApp03] requests the service [CMService04]. The [CMApp01] stops offering service [CMService03]. All services are supposed to be found once available. If a service is not available, the requesting application is expected to have the possibility to assess the availability. Note: As for order of offering, no particular order of offering and requesting is necessary.		
Pre-conditions	- [CM Tester] is connected to both ECUs. - Both ECUs are in Machine State Parking. - [CMApp01] on [ECU1] and [CMApp02], [CMApp03] on [ECU2] are shut down according to Machine State.		
Post-conditions	CM Tester is disconnected to both ECUs.		
Main Test Execution			
Test Steps			**Pass Criteria**
Step 1	[CM Tester] Request change of Machine State to Driving for [ECU2].		Machine State for [ECU2] is changed to Driving.
Step 2	[CMApp02] Offer service [CMService01].		

▽

△

Step 3	[CMApp02] Offer service [CMService02].	
Step 4	[CMApp03] Request service [CMService02].	Service discovery callback with a handle for service [CMService02] is received by [CMApp03].
Step 5	[CM Tester] Trigger Application [CMApp02] to Stop Offering service [CMService02].	
Step 6	[CMApp02] Offer service [CMService02].	Service discovery callback with a handle for service [CMService02] is received by [CMApp03].
Step 7	[CMApp02] Request service [CMService03].	Service is not available.
Step 8	[CM Tester] Request change of Machine State to Driving for [ECU1].	Machine State for [ECU1] is changed to Driving.
Step 9	[CMApp01] Offer service [CMService03].	
Step 10	[CMApp02] Request service [CMService03].	Service discovery callback with a handle for service [CMService03] received by [CMApp02].
Step 11	[CMApp01] Request service [CMService01].	Service discovery callback with a handle for service [CMService01] is received by [CMApp01].
Step 12	[CMApp03] Request service [CMService04].	Service is not available.
Step 13	[CMApp01] Stop offering service [CMService03].	
Step 14	[CMApp02] Request service [CMService03]	Service is not available.

4.2.2 [STS_CM_00002] Communication for Methods.

Test Objective	To verify that the applications are able to offer, request and receive services and that communication work in a one-to-n communication topology for Methods.		
ID	STS_CM_00002	**State**	Draft
Affected Functional Cluster	Communication Management		
Trace to RS Criteria	[RS_CM_00101], [RS_CM_00102], [RS_CM_00211], [RS_CM_00212], [RS_CM_00213], [RS_CM_00214], [RS_CM_00215], [RS_CM_00225]		
Reference to Test Environment	STC_CM_00002 in Test configurations Communication Management		

▽

△

Configuration Parameters	- The existing communication services comprise the following (service names are arbitrary): - [CMService05]: Offered by [CMApp04], requested by [CMApp05]. - [CMService06]: Offered by [CMApp02], requested by [CMApp04]. - [CMService07]: Offered by [CMApp03], requested by [CMApp04]. - [CMService05] service receives requested services synchronously. - [CMService06] service receives requested services asynchronously. One by querying applications and another by triggering applications. - [CMService07] service is an attribute for fire & forget methods.	
Summary	Firstly the [CMApp04] application on [ECU1] offers the service [CMService05]. This service is requested by one [CMApp05] instance on [ECU2] and another [CMApp05] instance on [ECU1]. The [CMApp02] application on [ECU2] offers the service [CMService06]. This service is requested by one [CMApp04] instance on [ECU1]. The [CMApp05] on [ECU2] receives data over service [CMService05] from [CMApp04] as synchronous service call. The [CMApp05] on [ECU1] receives data over service [CMService05] from [CMApp04] as synchronous service call. The [CMApp04] receives data as asynchronous service call by querying application [CMApp02] over service [CMService06]. Then [CMApp04] again request service [CMService06]. The [CMApp03] application on [ECU2] offers service [CMService07]. This service is requested by one [CMApp04] instance on [ECU1] as fire & forget service call. Then [CMApp04] receives data over service [CMService06] from [CMApp02] as asynchronous service call by notification. Through successful service discovery, a one-to-n communication topology is established. Note: As for order of offering, no particular order of offering and requesting is necessary.	
Pre-conditions	- [CM Tester] is connected to both ECUs. - Both ECUs are in Machine State Parking. - [CMApp04], [CMApp05] on [ECU1] and [CMApp02], [CMApp03], [CMApp05] on [ECU2] are shut down according to Machine State.	
Post-conditions	CM Tester is disconnected to both ECUs.	
Main Test Execution		
Test Steps		**Pass Criteria**
Step 1	[CMApp04] Offer service [CMService05].	
Step 2	[CMApp05] [ECU2] Request service [CMService05].	Service discovery callback with a handle for service [CMService05] is received by [CMApp05] [ECU2].
Step 3	[CMApp05] [ECU1] Request service [CMService05].	Service discovery callback with a handle for service [CMService05] is received by [CMApp05] [ECU1].
Step 4	[CMApp02] Offer service [CMService06].	
Step 5	[CMApp04] Request service [CMService06].	Service discovery callback with a handle for service [CMService06] is received by [CMApp04] [ECU1].
Step 6	[CMApp05] [ECU2] Receive vehicle data over service [CMService05] from [CMApp04].	[CMApp05] [ECU2] Data is received from [CMApp04] over service [CMService05].

▽

△

Step 7	[CMApp05] [ECU1] Receive vehicle data over service [CMService05] from [CMApp04].	[CMApp05] [ECU1] Data is received from [CMApp04] over service [CMService05].
Step 8	[CMApp04] Receive vehicle data over service [CMService06].	[CMApp04] Data is received over service [CMService06] by querying application [CMApp02]
Step 9	[CMApp04] Request service [CMService06].	Service discovery callback with a handle for service [CMService06] is received by [CMApp04] [ECU1].
Step 10	[CMApp03] Offer service [CMService07].	
Step 11	[CMApp04] Request service [CMService07] by fire & forget methods.	Service discovery callback with a handle for service [CMService07] may or may not be received by [CMApp04] [ECU1].
Step 12	[CMApp04] Receive vehicle data over service [CMService06].	[CMApp04] is notified that the result is available and can be received from application [CMApp04] over service [CMService06].

4.2.3 [STS_CM_00003] Communication for Events based on polling-based style.

Test Objective	To verify that the applications are able to offer, subscribe, receive and stop subscribing services and that communication work in a one-to-n communication topology for Events. The applications are able to receive events and access them in polling-based style.		
ID	STS_CM_00003	**State**	Draft
Affected Functional Cluster	Communication Management		
Trace to RS Criteria	[RS_CM_00101], [RS_CM_00102], [RS_CM_00104], [RS_CM_00105], [RS_CM_00106], [RS_CM_00201], [RS_CM_00202], [RS_CM_00206]		
Reference to Test Environment	STC_CM_00002 in Test configurations Communication Management		
Configuration Parameters	- The existing communication services comprise the following (service names are arbitrary): - [CMService08]: Offered by [CMApp04], requested by [CMApp05]. - Service [CMService08] is an attribute of Events. - Reception of services from Server to Proxy is possible using pooling-based style.		
Summary	First [CM Tester] request applications on [ECU1] and [ECU2] to change Machine State to Driving. [CM Tester] Request extended diagnostic session on [ECU1] and [ECU2] [CM Tester] trigger application [CMApp04] [ECU2] to start offering service [CMService08] and then application [CMApp04][ECU2]or[ECU1] start offering service [CMService08]. Service [CMService08] is subscribed by application [CMApp05] instance on [ECU1]. The application [CMApp05] [ECU1] Queue received events, <n> being the queue length. ▽		

▽

△

	△ Service [CMService08] is subscribed by application [CMApp05] instance on [ECU2]. The application [CMApp05] [ECU2] Queue received events, <n> being the queue length. The application [CMApp05] [ECU1] monitors state of subscription, which is offered by [CMApp04] of service [CMService08]. The application [CMApp05] [ECU2] monitors state of subscription, which is offered by [CMApp04] of service [CMService08]. [CM Tester] will trigger application [CMApp04] [ECU1] to start sending service [CMService08]. The application [CMApp04] [ECU2] will send service event over service [CMService08]. The application [CMApp05] [ECU2] poll for receiving events from application [CMApp04] over service [CMService08]. The application [CMApp05] [ECU1] poll for receiving events from application [CMApp04] over service [CMService08]. [CM Tester] trigger application [CMApp05] [ECU2] and application [CMApp05] [ECU1] to stop subscribing service [CMService08]. The application [CMApp05] [ECU2] Monitor state of subscription from service [CMService08] of application [CMApp04]. The application [CMApp05] [ECU1] Monitor state of subscription from service [CMService08] of application [CMApp04]. Through successful service discovery, a one-to-n communication topology is established. Note: As for order of offering, no particular order of offering and requesting is necessary.	
Pre-conditions	- [CM Tester] is connected to both ECUs. - Both ECUs are in Machine State Parking. - [CMApp04], [CMApp05] on [ECU2] and [CMApp05] on [ECU1] are shut down according to Machine State.	
Post-conditions	CM Tester is disconnected to both ECUs.	
Main Test Execution		
Test Steps		**Pass Criteria**
Step 1	[CM Tester] Request change of Machine State to Driving for [ECU1] and [ECU2].	
Step 2	[CM Tester] Trigger Application [CMApp04][ECU2] to Start Offering service [CMService08].	
Step 3	[CMApp05][ECU1] Subscribe to service [CMService08].	
Step 4	[CMApp05] [ECU1] Queue received events, <n> being the queue length	
Step 5	[CMApp05][ECU2] Subscribe to service [CMService08].	
Step 6	[CMApp05] [ECU2] Queue received events, <n> being the queue length	
Step 7	[CMApp05][ECU1] Monitor state of subscription over service [CMService08].	[CMApp05] [ECU1] gets the current status of subscription and notification if it changes from service [CMService08] of application [CMApp04].

▽

△

Step 8	[CMApp05][ECU2] Monitor state of subscription over service [CMService08].	[CMApp05] [ECU2] gets the current status of subscription and notification if it changes from service [CMService08] of application [CMApp04].
Step 9	[CM Tester] Trigger Application [CMApp04][ECU2] to Start sending service [CMService08].	
Step 10	[CMApp04] [ECU2] send only 10 service event [CMService08]	
Step 11	[CMApp05] [ECU2] Poll for receiving events from application [CMApp04] over service [CMService08].	[CMApp05] [ECU2] Event is not received over service [CMService05] of application [CMApp04].
Step 12	[CMApp05] [ECU1] Poll for receiving events from application [CMApp04] over service [CMService08].	[CMApp05] [ECU1] Event is not received over service [CMService05] of application [CMApp04].
Step 13	[CM Tester] Trigger Application [CMApp05][ECU2] to Stop subscription of service [CMService08]	
Step 14	[CM Tester] Trigger Application [CMApp05][ECU1] to Stop subscription of service [CMService08]	
Step 15	[CMApp05] [ECU2] Monitor state of subscription from service [CMService08] of application [CMApp04].	[CMApp05] [ECU2] gets the current status of subscription, i.e. [CMApp05] [ECU2] has stopped subscription from service [CMService05].
Step 16	[CMApp05] [ECU1] Monitor state of subscription from service [CMService08] of application [CMApp04].	[CMApp05] [ECU1] gets the current status of subscription, i.e. [CMApp05] [ECU2] has stopped subscription from service [CMService05].

4.2.4 [STS_CM_00004] Communication for Events based on event-based style.

Test Objective	To verify that the applications are able to offer, subscribe, monitor, receive and stop subscribing services and that communication work in a one-to-n communication topology for Events. The applications are able to receive events and access them in event-based style.		
ID	STS_CM_00004	**State**	Draft
Affected Functional Cluster	Communication Management		
Trace to RS Criteria	[RS_CM_00101], [RS_CM_00102], [RS_CM_00104], [RS_CM_00105], [RS_CM_00106], [RS_CM_00201], [RS_CM_00203], [RS_CM_00206]		
Reference to Test Environment	STC_CM_00002 in Test configurations Communication Management		

▽

△

Configuration Parameters	- The existing communication services comprise the following (service names are arbitrary): - [CMService05]: Offered by [CMApp04], requested by [CMApp05]. - Service [CMService05] is an attribute of Events. - Reception of services from Server to Client is possible using event-based style.	
Summary	First [CM Tester] request applications on [ECU1] and [ECU2] to change Machine State to Driving. [CM Tester] Request extended diagnostic session [ECU1] and [ECU2]. [CM Tester] trigger application [CMApp04] [ECU1] to start offering service [CMService05] and then application [CMApp04][ECU1] start offering service [CMService05]. Service [CMService05] is subscribed by an application [CMApp05] instance on [ECU1]. The application [CMApp05] [ECU1] Queue received events, <n> being the queue length. Service [CMService05] is subscribed by another application [CMApp05] instance on [ECU2]. The application [CMApp05] [ECU2] Queue received events, <n> being the queue length. The application [CMApp05] [ECU2] monitors state of subscription, which is offered by [CMApp04] of service [CMService05]. The application [CMApp05] [ECU1] monitors state of subscription, which is offered by [CMApp04] of service [CMService05]. [CM Tester] will trigger application [CMApp04] [ECU1] to start sending service [CMService05]. The application [CMApp04] [ECU1] will send service event over service [CMService05]. [CMApp05] [ECU2] Get triggered when receiving event over service [CMService05] of application [CMApp04] [CMApp05] [ECU1] Get triggered when receiving event over service [CMService05] of application [CMApp04] [CM Tester] trigger application [CMApp05] [ECU2] and application [CMApp05] [ECU1] to stop subscribing service [CMService05]. [CMApp05] [ECU1] Monitor state of subscription from service [CMService05] of application [CMApp04]. [CMApp05] [ECU2] Monitor state of subscription from service [CMService05] of application [CMApp04]. Through successful service discovery, a one-to-n communication topology is established. Note: As for order of offering, no particular order of offering and requesting is necessary.	
Pre-conditions	- [CM Tester] is connected to both ECUs. - Both ECUs are in Machine State Parking. - [CMApp04], [CMApp05] on [ECU1] and [CMApp05] on [ECU2] are shut down according to Machine State.	
Post-conditions	CM Tester is disconnected to both ECUs.	
Main Test Execution		
Test Steps		**Pass Criteria**
Step 1	[CM Tester] Request change of Machine State to Driving for [ECU1] and [ECU2].	
Step 2	[CM Tester] Trigger Application [CMApp04][ECU1] to Start Offering service [CMService05].	
Step 3	[CMApp05] [ECU1] Subscribe to service [CMService05].	
Step 4	[CMApp05] [ECU1] Queue received events, <n> being the queue length.	

▽

△

Step 5	[CMApp05] [ECU2] Subscribe to service [CMService05].	
Step 6	[CMApp05] [ECU2] Queue received events, <n> being the queue length.	
Step 7	[CMApp05][ECU1] Monitor state of subscription over service [CMService05].	[CMApp05] [ECU1] gets the current status of subscription and notification if it changes from service [CMService05] of application [CMApp04].
Step 8	[CMApp05][ECU2] Monitor state of subscription over service [CMService05].	[CMApp05] [ECU2] gets the current status of subscription and notification if it changes from service [CMService05] of application [CMApp04].
Step 9	[CM Tester] Trigger Application [CMApp04][ECU2] to Start sending service [CMService05].	
Step 10	[CMApp04] [ECU1] send service event [CMService05].	
Step 11	[CMApp05] [ECU2] Get triggered when receiving event over service [CMService05] of application [CMApp04].	[CMApp05] [ECU2] Events received and read them at the same time from service [CMService05].
Step 12	[CMApp05] [ECU1] Get triggered when receiving event over service [CMService05].	[CMApp05] [ECU1] Events received and read them at the same time from service [CMService05] of application [CMApp04].
Step 13	[CM Tester] Trigger Application [CMApp05][ECU2] to Stop subscription of service [CMService05]	
Step 14	[CM Tester] Trigger Application [CMApp05][ECU1] to Stop subscription of service [CMService05]	
Step 15	[CMApp05] [ECU1] Monitor state of subscription from service [CMService05] of application [CMApp04].	[CMApp05] [ECU1] gets the current status of subscription, i.e.[CMApp05] [ECU1] has stopped the subscription from service [CMService05].
Step 16	[CMApp05] [ECU2] Monitor state of subscription from service [CMService05] of application [CMApp04].	[CMApp05] [ECU2] gets the current status of subscription, i.e.[CMApp05] [ECU2] has stopped the subscription from service [CMService05].

4.2.5 [STS_CM_00005] Communication for Fields.

Test Objective	To verify that the applications are able to query (get) and modify (set) field value and that communication work for Fields.		
ID	STS_CM_00005	**State**	Draft
Affected Functional Cluster	Communication Management		
Trace to RS Criteria	[RS_CM_00216], [RS_CM_00217], [RS_CM_00218], [RS_CM_00219], [RS_CM_00220], [RS_CM_00221]		
Reference to Test Environment	STC_CM_00001 in Test configurations Communication Management		
Configuration Parameters	- The existing communication services comprise the following (service names are arbitrary): - [CMService05]: Offered by [CMApp04], requested by [CMApp05].		
Summary	Initially [CM Tester] requests applications to change Machine State to Driving. [CM Tester] requests [CMApp05] to get the current field value of service [CMService05] [CMApp04]. In turn [CMApp05] requests [CMApp04] to get the current field value of service [CMService05] [CMApp04]. The [CMApp04] provides a method to get the current field value of service [CMService05] [CMApp04]. [CM Tester] requests [CMApp05] to set the current field value of service [CMService05] [CMApp04]. In turn [CMApp05] requests [CMApp04] to set the current field value of service [CMService05] [CMApp04]. The [CMApp04] provides a method to set the current field value of service [CMService05] [CMApp04]. [CMApp04] sends normal return code notification to [CMApp05]. [CMApp05] returns a normal return code to [CM Tester]. Note: As for order of offering, no particular order of offering and requesting is necessary.		
Pre-conditions	- [CM Tester] is connected to [CMApp05]. - Both ECUs are in Machine State Parking. - Through successful service discovery, a communication is established. - A field without a setter and without a getter shall not exist. - The field shall contain at least a getter or a setter.		
Post-conditions	CM Tester is disconnected from CMApp05. - [CMApp04] on [ECU1] and [CMApp05] on [ECU1] are shut down according to Machine State.		
Main Test Execution			
Test Steps		**Pass Criteria**	
Step 1	[CM Tester] Request change of Machine State to Driving.		
Step 2	[CM Tester] Request [CMApp05] to get the current field value of service [CMService05] [CMApp04].		
Step 3	[CMApp05] Request [CMApp04] to get the current field value of service [CMService05] [CMApp04].	[CMApp04] Receives the request from application [CMApp05].	
Step 4	[CMApp04] Provides a method to get the current field value of service [CMService05] [CMApp04].	[CMApp05] Receives response message from [CMApp04].	

▽

△

Step 5	[CMApp05] Returns the current field value of service [CMService05][CMApp04] to [CM Tester].	[CM Tester] Receives the default field value (e.g. zero) of [CMService05][CMApp04].
Step 6	[CM Tester] Request [CMApp05] to set the current field value of service [CMService05][CMApp04].	
Step 7	[CMApp05] Request [CMApp04] to set the field value of service [CMService05][CMApp04].	[CMApp04] Receives the request from application [CMApp05].
Step 8	[CMApp04] Provides a method to set the current field value of service [CMService05][CMApp04].	[CMApp05] Receives response message from [CMApp04].
Step 9	[CMApp04] sends normal response to [CMApp05].	[CMApp05] Receives response from[CMApp04].
Step 10	[CMApp05] returns a normal return code to CM tester	[CM Tester] Receives termination notification from[CMApp04].
Step 11	[CM Tester] Request [CMApp05] to get the set field value of service [CMService05][CMApp04].	
Step 12	[CMApp05] Request [CMApp04] to get the current field value of service [CMService05] [CMApp04].	[CMApp04] Receives the request from application [CMApp05].
Step 13	[CMApp04] Provides a method to get the current field value of service [CMService05] [CMApp04].	[CMApp05] Receives response message from [CMApp04].
Step 14	[CMApp05] Returns the set field value of service [CMService05][CMApp04] to [CM Tester].	[CM Tester] Receives the set field value (set in the previous steps) of [CMService05][CMApp04].

4.2.6 [STS_CM_00006] Communication for Field Notification.

Test Objective	To verify that the applications are able to receive notifications and that communication work for Fields.		
ID	STS_CM_00006	**State**	Draft
Affected Functional Cluster	Communication Management		
Trace to RS Criteria	[RS_CM_00216], [RS_CM_00217], [RS_CM_00218], [RS_CM_00219], [RS_CM_00220], [RS_CM_00221], [RS_CM_00226], [RS_CM_00227]		
Reference to Test Environment	STC_CM_00001 in Test configurations Communication Management		
Configuration Parameters	- The existing communication services comprise the following (service names are arbitrary): - [CMService05]: Offered by [CMApp04], requested by [CMApp05].		

▽

△

Summary	Initially [CM Tester] requests applications to change Machine State to Driving. [CM Tester] requests [CMApp05] to subscribe [FIELD1] event notification of service [CMService05][CMApp04]. In turn [CMApp05] requests [CMApp04] to subscribe [FIELD1] event notification of service [CMService05][CMApp04]. [CMApp04] sends normal return code of [FIELD1] event subscription to [CMApp05]. [CMApp05] returns a normal return code to [CM Tester]. [CM Tester] requests [CMApp05] to set value <x> (not default value) to [FIELD1] of service [CMService05][CMApp04]. In turn [CMApp05] requests [CMApp04] to set value <x> to [FIELD1] of service [CMService05][CMApp04]. [CMApp04] sends normal return code of setting [FIELD1] to [CMApp05]. [CMApp05] sends a normal return code to [CM Tester]. [CM Tester] receives normal return code. [CMApp04] sends event notification of changing [FIELD1] value. [CMApp05] receives event notification of changing [FIELD1] value. After a time <tx>, [CM Tester] requests [CMApp05] to confirm receiving event notification. [CMApp05] sends received event notifications to [CM Tester]. [CM Tester] receives event notification. Note: As for order of offering, no particular order of offering and requesting is necessary.	
Pre-conditions	- [CM Tester] is connected to [CMApp05]. - Both ECUs are in Machine State Parking. - Through successful service discovery, a communication is established. - A field without a notifier shall not exist. - The field shall contain at least one notifier.	
Post-conditions	CM Tester is disconnected from CMApp05. [CMApp04] and [CMApp05] are shut down according to Machine State.	
Main Test Execution		
Test Steps		**Pass Criteria**
Step 1	[CM Tester] Request change of Machine State to Driving.	
Step 2	[CM Tester] Requests [CMApp05] to subscribe [FIELD1] event notification of service [CMService05][CMApp04].	
Step 3	[CMApp05] Requests [CMApp04] to subscribe [FIELD1] event notification of service [CMService05][CMApp04].	[CMApp04] Receives the request from application [CMApp05].
Step 4	[CMApp04] Sends normal return code of [FIELD1] event subscription to [CMApp05].	[CMApp05] Receives response message from [CMApp04].
Step 5	[CMApp05] Returns a normal return code to [CM Tester].	[CM Tester] Receives the return code.
Step 6	[CM Tester] Requests [CMApp05] to set value <x> (not default value) to [FIELD1] of service [CMService05][CMApp04].	

▽

△

Step 7	[CMApp05] Requests [CMApp04] to set value <x> to [FIELD1] of service [CMService05][CMApp04].	[CMApp04] Receives the request from application [CMApp05].
Step 8	[CMApp04] Sends normal return code of setting [FIELD1] to [CMApp05].	[CMApp05] Receives response message from [CMApp04].
Step 9	[CMApp05] Sends a normal return code to [CM Tester].	[CM Tester] Receives the normal return code.
Step 10	[CMApp04] Sends event notification of changing [FIELD1] value.	[CMApp05] Receives event notification of changing [FIELD1] value.
Step 11	[CM Tester] After time <tx>, requests [CMApp05] to confirm receiving event notification.	
Step 12	[CMApp05] Sends received event notification to [CM Tester].	[CM Tester] Receives event notification.

4.2.7 [STS_CM_00007] Service discovery evaluating service contract version.

Test Objective	To verify whether service discovery can establish the communication path between applications by evaluating service version and black listed version.		
ID	STS_CM_00007	**State**	Draft
Affected Functional Cluster	Communication Management		
Trace to RS Criteria	[RS_CM_00700], [RS_CM_00701]		
Trace to SWS	[SWS_CM_99003], [SWS_CM_10202]		
Reference to Test Environment	STC_CM_00001 in Test configurations Communication Management		
Configuration Parameters	- The existing communication services comprise the following (service names are arbitrary): - [CMServiceA_V1_0] is offered by [CMApp01], requested by [CMApp02]. - [CMServiceA_V1_1] is offered by [CMApp01], requested by [CMApp03]. - [CMServiceA_V1_2] is offered by [CMApp03], requested by [CMApp02]. - [CMServiceA_V2_0] is offered by [CMApp01]. - [CMApp02] blacklisted version 1.2 in required instance i.e. [CMServiceA_V1_1]. - CMServiceA_V1_0: • Event_A - CMServiceA_V1_1: • Event_A • Event_B - CMServiceA_V1_2: • Event_A ▽		

▽

△

	△ • Event_B • Event_C - CMServiceA_V2_0: • Event_D	
Summary	[CMApp01] and [CMApp02] are on [ECU1] and [CMApp03] is on [ECU2]. [CMApp01] and [CMApp02] are started when machine state for [ECU1] changes to driving. [CMApp01] offers the service [CMServiceA_V1_0] and [CMApp02] request for the same. [CMApp03] is started when the machine state for [ECU2] changes to driving and requests the service [CMServiceA_V1_1]. Connection is established between [CMApp01 - CMApp02] and not between [CMApp01 - CMApp03]. • CMApp01 - CMApp02 (Exact match) • CMApp01 - CMApp03 (No matching service found) [CMApp01] stop offering the service [CMServiceA_V1_0] and offer service [CMServiceA_V1_1]. [CMApp02] and [CMApp03] again request for service [CMServiceA_V1_0] and [CMServiceA_V1_1] respectively. Connection is established between [CMApp01 - CMApp03] and not between [CMApp01 - CMApp02]. • CMApp01 - CMApp02 (CMServiceA_V1_1 is blacklisted) • CMApp01 - CMApp03 (Exact match) [CMApp03] offers the service [CMServiceA_V1_2] and [CMApp02] again request for service [CMServiceA_V1_0] Connection is established between [CMApp02-CMApp03] with service [CMServiceA_V1_2] (Backward compatibility with CMServiceA_V1_0). Note: All the steps will be triggered by CMTester and result will be sent back to CMTester.	
Pre-conditions	- [CM Tester] is connected to both ECUs. - Both ECUs are in Machine State Parking. - [CMApp01], [CMApp02] on [ECU1] and [CMApp03] on [ECU2] are shut down according to Machine State.	
Post-conditions	CM Tester is disconnected to both ECUs.	
Main Test Execution		
Test Steps		**Pass Criteria**
Step 1	[CMTester] Request machine state change to driving for [ECU1].	Machine state on [ECU1] changed to driving.
Step 2	[CMApp01] offer service CMServiceA_V1_0	
Step 3	[CMApp02] request service CMServiceA_V1_0	Service discovery callback with a handle for service [CMServiceA_V1_0] should be received by [CMApp02] (Exact match).
Step 4	[CMTester] Request machine state change to driving for [ECU2]	Machine state on [ECU2] changed to driving.
Step 5	[CMApp03] request service CMServiceA_V1_1	No matching service found
Step 6	[CMApp01] stop offering service [CMServiceA_V1_0].	
Step 7	[CMApp01] offer service [CMServiceA_V1_1]	
Step 8	[CMApp02] request service [CMServiceA_V1_0]	No matching service found (CMServiceA_V1_1 is blacklisted).
Step 9	[CMApp03] again request for service [CMServiceA_V1_1]	Service discovery callback with a handle for service [CMServiceA_V1_1] should be received by [CMApp03] (Exact match).
Step 10	[CMApp03] offer service [CMServiceA_V1_2].	

▽

△

Step 11	[CMApp02] request service [CMServiceA_V1_0].	Service discovery callback with a handle for service [CMServiceA_V1_2] should be received by [CMApp02] (Backward compatible with CMServiceA_V1_0).
Step 12	[CMApp01] stop offering service [CMServiceA_V1_1].	
Step 13	[CMApp03] stop offering service [CMServiceA_V1_2]	
Step 14	[CMApp01] offer service [CMServiceA_V2_0].	
Step 15	[CMApp03] request service [CMServiceA_V1_1].	No matching service found.

4.2.8 [STS_CM_00008] Service contract versioning for Event(event-based) communication.

Test Objective	To verify whether Communication Management supports service contract versioning for Event(event-based) communication.		
ID	STS_CM_00008	**State**	Draft
Affected Functional Cluster	Communication Management		
Trace to RS Criteria	[RS_CM_00500]		
Trace to SWS	[SWS_CM_99003], [SWS_CM_01010], [SWS_CM_09004]		
Reference to Test Environment	STC_CM_00002 in Test configurations Communication Management		
Configuration Parameters	- [CMServiceA_V1_0] is offered by [CMApp01], requested by [CMApp02] - [CMServiceA_V1_2] is offered by [CMApp03], requested by [CMApp02] - [CMServiceA_V2_0] is offered by [CMApp01] - CMServiceA_V1_0: • Event_A - CMServiceA_V1_2: • Event_A • Event_B • Event_C - CMServiceA_V2_0: • Event_D		
Summary	[CMApp01] and [CMApp02] are on [ECU1] and [CMApp03] is on [ECU2]. [CMApp01] and [CMApp02] are started when machine state for [ECU1] changes to driving. [CMApp01] offers the service [CMServiceA_V1_0]. [CMApp02] request and subscribe to service [CMServiceA_V1_0] and receives the events from [CMApp01]. [CMApp02] stop find service [CMServiceA_V1_0]. [CMApp02] request for service [CMServiceA_V1_2]. [CMApp02] matching service not found [CMServiceA_V1_2]. ▽		

▽

△

	△ [CMApp03] is started when the machine state for [ECU2] changes to driving and offer service [CMServiceA_V1_2]. [CMApp02] request for service [CMServiceA_V1_0] and subscribe to received service [CMServiceA_V1_2]. Note: All the steps will be triggered by CMTester and result will be sent back to CMTester.	
Pre-conditions	- [CM Tester] is connected to both ECUs. - Both ECUs are in Machine State Parking. - [CMApp01], [CMApp02] on [ECU1] and [CMApp03] on [ECU2] are shut down according to Machine State..	
Post-conditions	CM Tester is disconnected to both ECUs.	
Main Test Execution		
Test Steps		**Pass Criteria**
Step 1	[CMTester] Request machine state change to driving for [ECU1]	Machine state on [ECU1] changed to driving.
Step 2	[CMApp01] offer service CMServiceA_V1_0	
Step 3	[CMApp02] request service CMServiceA_V1_0	Service discovery callback with a handle for service [CMServiceA_V1_0] should be received by [CMApp02] (Exact match).
Step 4	[CMApp02] subscribe to service [CMServiceA_V1_0]	
Step 5	[CMApp02] Get the state of subscription for service [CMServiceA_V1_0]	State should be kSubscribed.
Step 6	[CMTester] Trigger application [CMApp01] to start sending the event over service [CMServiceA_V1_0].	
Step 7	[CMApp02] Get triggered when receiving events from application [CMApp01] over service [CMServiceA_V1_0].	[CMApp02] should receive the event data from [CMApp01] over service [CMServiceA_V1_0].
Step 8	[CMApp02] stop find service [CMServiceA_V1_0].	
Step 9	[CMApp02] request service [CMServiceA_V1_2].	No matching service found
Step 10	[CMTester] Request machine state change to driving for [ECU2]	Machine state on [ECU2] changed to driving.
Step 11	[CMApp03] offer service [CMServiceA_V1_2].	
Step 12	[CMApp01] stop offering service [CMServiceA_V1_0].	
Step 13	[CMApp02] request service [CMServiceA_1_0].	Service discovery callback with a handle for service [CMServiceA_V1_2] should be received by [CMApp02] (Backward compatible with CMServiceA_V1_0).
Step 14	[CMApp02] subscribe and set receive handler to service [CMServiceA_V1_2].	
Step 15	[CMApp02] Get the state of subscription for service [CMServiceA_V1_2].	State should be kSubscribed.
Step 16	[CMTester] Trigger application [CMApp03] to start sending the event over service [CMServiceA_V1_2].	
Step 17	[CMApp02] Get triggered when receiving events from application [CMApp03] over service [CMServiceA_V1_2].	[CMApp02] should receive the event data from [CMApp03] over service [CMServiceA_V1_2].

4.2.9 [STS_CM_00009] Service contract versioning for Method communication.

Test Objective	To verify whether Communication Management supports service contract versioning for Method.		
ID	STS_CM_00009	**State**	Draft
Affected Functional Cluster	Communication Management		
Trace to RS Criteria	[RS_CM_00500], [RS_CM_00501]		
Trace to SWS	[SWS_CM_99003], [SWS_CM_01010], [SWS_CM_09004]		
Reference to Test Environment	STC_CM_00002 in Test configurations Communication Management		
Configuration Parameters	- [CMServiceB_V1_0] is offered by [CMApp02], requested by [CMApp01] - [CMServiceB_V1_1] is offered by [CMApp02], requested by [CMApp03] - [CMServiceB_V2_0] is offered by [CMApp02] - CMServiceB_V1_0: • Method_A - CMServiceB_V1_1: • Method_A • Method_B - CMServiceB_V2_0: • Method_C		
Summary	[CMApp01] and [CMApp02] are on [ECU1] and [CMApp03] is on [ECU2]. [CMApp01] and [CMApp02] are started when machine state for [ECU1] changes to driving [CMApp02] offers the service [CMServiceB_V1_0]. [CMApp01] request for service [CMServiceB_V1_0]. [CMApp01] receives data from [CMApp02] over [CMServiceB_V1_0] as synchronous service call [CMApp03] is started when the machine state for [ECU2] changes to driving and request for service [CMServiceB_V1_1]. [CMApp03] matching service not found. [CMApp02] stop offering the service [CMServiceB_V1_0] and offer service [CMServiceB_V1_1]. [CMApp01] and [CMApp03] again request for service [CMServiceB_V1_0] and [CMServiceB_V1_1] respectively. Connection is established between [CMApp01] - [CMApp02] and [CMApp02] - [CMApp03] over service [CMServiceB_V1_1]. • CMApp01 - CMApp02 (Backward compatible with [CMServiceB_V1_0]) • CMApp02 - CMApp03 (Exact match) [CMApp01] receives data from [CMApp02] over [CMServiceB_V1_1] as synchronous service call. [CMApp03] receives data from [CMApp02] over [CMServiceB_V1_1] as synchronous service call. Note: All the steps will be triggered by CMTester and result will be sent back to CMTester.		
Pre-conditions	- [CM Tester] is connected to both ECUs. - Both ECUs are in Machine State Parking. - [CMApp01], [CMApp02] on [ECU1] and [CMApp03] on [ECU2] are shut down according to Machine State.		
Post-conditions	CM Tester is disconnected to both ECUs.		
Main Test Execution			
Test Steps			**Pass Criteria**

▽

△

Step 1	[CMTester] Request machine state change to driving for [ECU1]	Machine state on [ECU1] changed to driving.
Step 2	[CMApp02] offer service [CMServiceB_V1_0]	
Step 3	[CMApp01] request service [CMServiceB_V1_0]	Service discovery callback with a handle for service [CMServiceB_V1_0] should be received by [CMApp01] (Exact match).
Step 4	[CMApp01] receive the data from [CMApp02] by calling Method_A over [CMServiceB_V1_0]	[CMApp01] should receive data from [CMApp02] over service [CMServiceB_V1_0].
Step 5	[CMTester] Request machine state change to driving for [ECU2]	Machine state on [ECU2] changed to driving.
Step 6	[CMApp03] request service [CMServiceB_V1_1].	No matching service found.
Step 7	[CMApp02] stop offering service [CMServiceB_V1_0].	
Step 8	[CMApp02] offer service [CMServiceB_V1_1]	
Step 9	[CMApp01] request service [CMServiceB_V1_0]	Service discovery callback with a handle for service [CMServiceB_V1_1] should be received by [CMApp01] (Backward compatible with [CMServiceB_V1_0]).
Step 10	[CMApp01] receive the data from [CMApp02] by calling Method_A over [CMServiceB_V1_1]	[CMApp01] should receive data from [CMApp02] over service [CMServiceB_V1_1].
Step 11	[CMApp03] again request service [CMServiceB_V1_1]	Service discovery callback with a handle for service [CMServiceB_V1_1] should be received by [CMApp03] (Exact match).
Step 12	[CMApp03] receive the data from [CMApp02] over [CMServiceB_V1_1]	[CMApp03] should receive data from [CMApp02] over service [CMServiceB_V1_1].

4.2.10 [STS_CM_00010] Service contract versioning for Field communication.

Test Objective	To verify whether Communication Management supports service contract versioning for Field communication.		
ID	STS_CM_00010	**State**	Draft
Affected Functional Cluster	Communication Management		
Trace to RS Criteria	[RS_CM_00500], [RS_CM_00501]		
Trace to SWS	[SWS_CM_99003], [SWS_CM_01010], [SWS_CM_09004]		
Reference to Test Environment	STC_CM_00001 in Test configurations Communication Management		

▽

△

Configuration Parameters	- [CMServiceC_V1_0] is offered by [CMApp03], requested by [CMApp01] - [CMServiceC_V1_1] is offered by [CMApp03], requested by [CMApp02] - [CMServiceC_V2_0] is offered by [CMApp03] - CMServiceC_V1_0: • Field_A - CMServiceB_V1_1: • Field_A • Field_B - CMServiceB_V2_0: • Field_C	
Summary	[CMApp01] and [CMApp02] are on [ECU1] and [CMApp03] is on [ECU2]. [CMApp01] and [CMApp02] are started when machine state for [ECU1] changes to driving. [CMApp03] is started when the machine state for [ECU2] changes to driving. [CMApp03] offers the service [CMServiceC_V1_0]. [CMApp01] request for service [CMServiceC_V1_0]. [CMApp01] subscribe to service [CMServiceC_V1_0]. [CMApp01] get the current field value from [CMApp03] over [CMServiceC_V1_0]. [CMApp03] update the field value of [CMServiceC_V1_0]. [CMApp01] receives the notification over service [CMServiceC_V1_0]. [CMApp02] request for service [CMServiceC_V1_1]. [CMApp02] matching service not found. [CMApp03] stop offering the service [CMServiceC_V1_0] and offer service [CMServiceC_V1_1]. [CMApp01] and [CMApp02] again request for service [CMServiceC_V1_0] and [CMServiceC_V1_1] respectively. Connection is established between [CMApp01] - [CMApp03] and [CMApp02] - [CMApp03] over service [CMServiceC_V1_1]. • CMApp01 - CMApp03 (backward compatible with CMServiceC_V1_0) • CMApp02 - CMApp03 (Exact match) [CMApp01] and [CMApp02] subscribe to service [CMServiceC_V1_1]. [CMApp01] sets the field value of [CMApp03] over service [CMServiceC_V1_1]. [CMApp02] gets the field value from [CMApp03] over [CMServiceC_V1_1]. [CMApp03] updates the field value. [CMApp01] and [CMApp02] receives the notification from [CMApp03] over service [CMServiceC_V1_1]. Note: All the steps will be triggered by CMTester and result will be sent back to CMTester.	
Pre-conditions	- [CM Tester] is connected to both ECUs. - Both ECUs are in Machine State Parking. - [CMApp01], [CMApp02] on [ECU1] and [CMApp03] on [ECU2] are shut down according to Machine State.	
Post-conditions	CM Tester is disconnected to both ECUs.	
Main Test Execution		
Test Steps		**Pass Criteria**
Step 1	[CMTester] Request machine state change to driving for [ECU1]	Machine state on [ECU1] changed to driving.
Step 2	[CMApp03] register the get and set handler for [CMServiceC_V1_0]	

▽

△

Step 3	[CMApp03] offer service CMServiceC_V1_0	
Step 4	[CMApp01] request service CMServiceC_V1_0	Service discovery callback with a handle for service [CMServiceC_V1_0] should be received by [CMApp01] (Exact match).
Step 5	[CMApp01] subscribe to service [CMServiceC_V1_0]	
Step 6	[CMApp01] get the field value over [CMServiceC_V1_0].	Default field value should be received by [CMApp01].
Step 7	[CMApp03] update the field value of [CMServiceC_V1_0]	[CMApp01] should receive the notification over service [CMServiceC_V1_0]
Step 8	[CMApp02] request service [CMServiceC_V1_1]	No matching service found.
Step 9	[CMApp03] stop offering service [CMServiceC_V1_0]	
Step 10	[CMApp03] register the get and set handler for [CMServiceC_V1_1]	
Step 11	[CMApp03] offer service [CMServiceC_V1_1]	
Step 12	[CMApp01] request service [CMServiceC_V1_0]	Service discovery callback with a handle for service [CMServiceC_V1_1] should be received by [CMApp01] (Backward compatible with CMServiceC_V1_0).
Step 13	[CMApp02] request service [CMServiceC_V1_1]	Service discovery callback with a handle for service [CMServiceC_V1_1] should be received by [CMApp02] (Exact match).
Step 14	[CMApp01] and [CMApp02] subscribe to service [CMServiceC_V1_1]	
Step 15	[CMApp01] set the field value of [CMApp03] over service [CMServiceC_V1_1]	
Step 16	[CMApp02] get the field value from [CMApp03] over [CMServiceC_V1_1]	[CMApp02] should receive the field value from [CMApp03] over service [CMServiceC_V1_1].
Step 17	[CMApp03] update the field value of service [CMServiceC_V1_1]	[CMApp01] and [CMApp02] should receive the notification from [CMApp03] over service [CMServiceC_V1_1].

4.3 Test cases Signal-To-Service

4.3.1 [STS_S2S_00001] Signal-To-Service Translation for Event(Incoming signal).

Test Objective	To verify whether application on Classic Platform and Adaptive Platform are able to perform event communication using Signal-To-Service Translation (Incoming signal).		
ID	STS_S2S_00001	**State**	Draft
Affected Functional Cluster	Communication Management		
Trace to RS Criteria	[RS_CM_00004]		

▽

△

Reference to Test Environment	STC_S2S_00001 in Test configurations SignalToService
Configuration Parameters	- Incoming signal from application on Classic Platform to application [CMApp01] on Adaptive Platform. - [CMService4] is offered by [S2S_Translator], requested by [CMApp01] - CMService4: • Event_A • Event_B • Event_C - ISignalIPduGroup: • ISignal0(uint8) - Pdu0 • ISignalGroup1 - Pdu1 ISignal1(uint8) ISignal2(uint16) • Direction - Out - Mapping: • Event_A - ISignal0 • Event_B - ISignal1 • Event_C - ISignal2
Summary	[S2S_Translator] and [CMApp01] are on [ECU1-Adaptive Platform]. [CMApp01] is started when the machine state for [ECU1] changes to Driving. First on Classic Platform ECU the state of ISignalIPduGroup is active. [S2S_Translator] offers the service [CMService4] and [CMApp01] request for the same. Connection is established between [S2S_Translator-CMApp01]. Applicaton on Classic Platform ECU sends the Pdu0 on the CAN channel which is forwarded on ethernet by GatewayECU. [S2S_Translator] send the Event_A with data received in ISignal0 and [CMApp01] receives the Event_A. Applicaton on Classic Platform ECU sends the Pdu1 on the CAN which is forwarded on ethernet by GatewayECU. [S2S_Translator] send the Event_B and Event_C with data received in ISignal1 and ISignal2 respectively and [CMApp01] receives the Event_B and Event_C. On Classic Platform ECU ISignalIPduGroup changes to inactive. [S2S_Translator] stop offering service [CMService4].
Pre-conditions	- [CM Tester] is connected to both ECUs. - ECU2, GatewayECU: Classic Platform and ECU1: Adaptive Platform. - [S2S_Translator] and [CMApp01] are on ECU1. - ECU1 is in machine state Parking. - Connections: ECU2 - GatewayECU: CAN GatewayECU - ECU1: Ethernet - CAN communication channel should be in COMM_FULL_COMMUNICATION state.
Post-conditions	CM Tester is disconnected to both ECUs.
Main Test Execution	

Test Steps		**Role of S2S Translator**	**Pass Criteria**
Step 1	[CMTester] Request machine state change to Driving for [ECU1]		Machine state on [ECU1] changed to Driving.

▽

△

Step 2	Change the status ISignalIPduGroup to active.	S2S_Translator should offer the service CMService4.	
Step 3	[CMApp01] request for the service [CMService4]		Service discovery callback with a handle for service [CMService4] should be received by [CMApp01].
Step 4	[CMApp01] subscribe for the service [CMService4]		
Step 5	Application on Classic Platform ECU sends the Pdu0 on the CAN channel which is forwarded on ethernet by GatewayECU.	[S2S_Translator] send the Event_A with data received in ISignal0.	[CMApp01] should receive the Event_A with data in ISignal0.
Step 6	Application on Classic Platform ECU sends the Pdu1 on the CAN channel which is forwarded on ethernet by GatewayECU.	[S2S_Translator] send the Event_B and Event_C with data received in ISignal1 and ISignal2 respectively.	[CMApp01] should receive the Event_B and Event_C with data in ISigna1 and ISignal2 respectively.
Step 7	Change the state of IsignalIPduGroup to inactive.	[S2S_Translator] should stop offering the service [CMService4].	
Step 8	[CMApp01] request service [CMService4]		No matching service found.

4.3.2 [STS_S2S_00002] Signal-To-Service Translation for Event(Outgoing signal).

Test Objective	To verify whether application on Classic Platform and Adaptive Platform are able to perform event communication using Signal-To-Service Translation (outgoing signal).		
ID	STS_S2S_00002	**State**	Draft
Affected Functional Cluster	Communication Management		
Trace to RS Criteria	[RS_CM_00004]		
Reference to Test Environment	STC_S2S_00001 in Test configurations SignalToService		
Configuration Parameters	- Outgoing signal from application on Adaptive Platform to application on Classic Platform. - [CMService5] is offered by [CMApp01], requested by [S2S_Translator] - CMService5: • Event_A • Event_B - ISignalIPduGroup: • ISignal0(uint8) - Pdu0 • ISignal1(uint16) - Pdu1 • Direction - In - Mapping: • Event_A - ISignal0 • Event_B - ISignal1		

▽

△

Summary	[S2S_Translator] and [CMApp01] are on [ECU1-Adaptive Platform]. [CMApp01] offers the service [CMService5]. On Classic Platform ECU Changes the state of ISignalIPduGroup to active. [S2S_Translator] request for the service [CMService5]. Connection is established between [CMApp01-S2S_Translator]. [S2S_Translator] subscribe the service [CMService5]. [CMApp01] sends the Event_A. [S2S_Translator] sends the Pdu0 to Application on Classic Platform ECU. [CMApp01] send the Event_B. [S2S_Translator] sends the Pdu1 to [CMApp01]. On Classic Platform ECU Changes the state of ISignalIPduGroup to inactive. [S2S_Translator] stop finding the service. [CMApp01] stop offering the service [CMService5].		
Pre-conditions	- [CM Tester] is connected to both ECUs. - ECU2, GatewayECU: Classic Platform and ECU1: Adaptive Platform. - [S2S_Translator] and [CMApp01] are on ECU1. - ECU1 is in machine state Parking. - Connections: ECU2 - GatewayECU: CAN GatewayECU - ECU1: Ethernet - CAN communication channel should be in COMM_FULL_COMMUNICATION state.		
Post-conditions	CM Tester is disconnected to both ECUs.		
Main Test Execution			
Test Steps		**Role of S2S Translator**	**Pass Criteria**
Step 1	[CMTester] Request machine state change to Driving for [ECU1]		Machine state on [ECU1] changed to Driving.
Step 2	[CMApp01] offer the service [CMService5]		
Step 3	Change the status ISignalIPduGroup to active.	S2S_Translator should request for the service CMService5.	Service discovery callback with a handle for service [CMService5] should be received by [S2S_Translator].
Step 4	[S2S_Translator] subscribe for the service [CMService5]		
Step 5	[CMApp01] send the Event_A.	[S2S_Translator] should send the Pdu0 to Application on Classic Platform ECU.	Application on Classic Platform ECU should receive the ISignal0.
Step 6	[CMApp01] send the Event_B.	[S2S_Translator] should send the Pdu1 to Application on Classic Platform ECU.	Application on Classic Platform ECU should receive the ISigna1.
Step 7	Change the state of ISignalIPduGroup to inactive.		
Step 8	[CMApp01] stop offering the service [CMService5]		

4.4 Test cases DDS

4.4.1 [STS_DDS_00001] Service discovery using DDS binding.

Test Objective	To verify the service discovery using DDS binding.		
ID	STS_DDS_00001	**State**	Draft
Affected Functional Cluster	Communication Management		
Trace to RS Criteria	[RS_CM_00101], [RS_CM_00102], [RS_CM_00105]		
Reference to Test Environment	STC_CM_00001 in Test configurations Communication Management		
Configuration Parameters	- [DDSService01] is offered by [DDSApp01], requested by [DDSApp02]. - [DDSService02] is offered by [DDSApp01], requested by [DDSApp03]. - The communication services comprise the following (names are arbitrary): - Service: • DDSService01: – Event_A – Event_B • DDSService02: – Event_C - Deployment: • DdsServiceInterfaceDeployment Note: The service and event names are arbitrary.		
Summary	[DDSApp01] and [DDSApp02] are on [ECU1], [DDSApp03] is on [ECU2]. [DDSApp01], [DDSApp02], [DDSApp03] are started when the machine state for [ECU1] and [ECU2] changes to Driving. [DDSApp01] offers the service [DDSService01]. [DDSApp02] requests for [DDSService01]. Connection is established between [DDSApp01] and [DDSApp02]. [DDSApp03] requests for [DDSService02]. Connection is not established as service is not available. [DDSApp01] stops offering the service [DDSService01] and offers [DDSService02]. [DDSApp03] requests for [DDSService02]. Connection is established between [DDSApp01] and [DDSApp03]. [DDSApp02] requests for [DDSService02]. Connection is not established as service is not available. [DDSApp01] stops offering the service [DDSService02]. Note: All the steps will be triggered by DDSTester and the result will be sent back to it.		
Pre-conditions	- [DDSTester] is connected to both ECU. - Both the ECUs are in machine state Parking.		
Post-conditions	[DDSTester] is disconnected from both ECUs.		
Main Test Execution			
Test Steps		**Pass Criteria**	

▽

△

Step 1	[DDSTester] request the machine state change to Driving on [ECU1] and [ECU2].	Machine state on [ECU1] and [ECU2] changed to Driving.
Step 2	[DDSApp01] offer the service [DDSService01].	
Step 3	[DDSApp02] request for the service [DDSService01].	Service discovery callback with a handle for service [DDSService01] should be received by [DDSApp02].
Step 4	[DDSApp03] request for the service [DDSService02].	Service not available.
Step 5	[DDSApp01] Stop offer service [DDSService01].	
Step 6	[DDSApp01] offer service [DDSService02].	
Step 7	[DDSApp03] request for the service [DDSService02].	Service discovery callback with a handle for service [DDSService02] should be received by [DDSApp03].
Step 8	[DDSApp02] request for the service [DDSService01].	Service not available.
Step 9	[DDSApp01] Stop offer service [DDSService02].	

4.4.2 [STS_DDS_00002] Event communication using DDS binding (event based).

Test Objective	To verify the event communication using DDS deployment (event based).		
ID	STS_DDS_00002	**State**	Draft
Affected Functional Cluster	Communication Management		
Trace to RS Criteria	[RS_CM_00101], [RS_CM_00102], [RS_CM_00103], [RS_CM_00104], [RS_CM_00105], [RS_CM_00106], [RS_CM_00201], [RS_CM_00203]		
Reference to Test Environment	STC_CM_00001 in Test configurations Communication Management		
Configuration Parameters	- [DDSService01] is offered by [DDSApp01], requested by [DDSApp02]. - [DDSService02] is offered by [DDSApp01], requested by [DDSApp03]. - Service: • DDSService01: – Event_A – Event_B • DDSService02: – Event_C - Deployment: • DdsServiceInterfaceDeployment. - Instance: • DdsProvidedServiceInstance • DdsRequiredServiceInstance Note: The service and event names are arbitrary.		
Summary	[DDSApp01] and [DDSApp02] are on [ECU1] and [DDSApp03] is on [ECU2]. [DDSApp01], [DDSApp02] and [DDSApp03] are started when the machine state for [ECU1] and [ECU2] changes to Driving. [DDSApp01] offers the service [DDSService01] and [DDSApp02] request for the same. ▽		

▽

△

	△ [DDSApp02] subscribes for [DDSService01]. [DDSApp01] sends the Event_A. [DDSApp02] registered EventReceiveHandler for Event_A gets triggered with the data sent by [DDSApp01]. [DDSApp01] sends the Event_B. [DDSApp02] registered EventReceiveHandler for Event_B gets triggered with the data sent by [DDSApp01]. [DDSApp01] stop offering the [DDSService01] and offer [DDSService02]. [DDSApp03] requests and subscribes for [DDSService02]. [DDSApp01] sends the Event_C. [DDSApp03] registered EventReceiveHandler for Event_C gets triggered with the data sent by [DDSApp01]. [DDSApp01] stops offering the service [DDSService02]. Note: All the steps will be triggered by DDSTester and the result will be sent back to it.	
Pre-conditions	- [DDSTester] is connected to both ECU. - Both the ECUs are in machine state Parking.	
Post-conditions	[DDSTester] is disconnected from both ECUs.	
Main Test Execution		
Test Steps		**Pass Criteria**
Step 1	[DDSTester] request the machine state change to Driving on [ECU1] and [ECU2].	Machine state on [ECU1] and [ECU2] changed to Driving.
Step 2	[DDSApp01] offer the service [DDSService01].	
Step 3	[DDSApp02] request for the service [DDSService01].	Service discovery callback with a handle for service [DDSService01] should be received by [DDSApp02].
Step 4	[DDSApp02] registers the EventReceiveHandler for Event_A and Event_B.	
Step 5	[DDSApp02] subscribe for the service [DDSService01].	
Step 6	[DDSApp02] Monitor state of subscription over service [DDSService01].	[DDSApp02] should receive the status as KSubscribed.
Step 7	[DDSApp01] send the Event_A.	Registered EventReceiveHandler should get triggered for Event_A in [DDSApp02] with data sent by [DDSApp01].
Step 8	[DDSApp01] send the Event_B.	Registered EventReceiveHandler should get triggered for Event_B in [DDSApp02] with data sent by [DDSApp01].
Step 9	[DDSApp02] Unsubscribe for the service [DDSService01].	
Step 10	[DDSApp02] Monitor state of subscription over service [DDSService01].	[DDSApp02] should receive the status as kNotSubscribed.
Step 11	[DDSApp01] stop offering the service [DDSService01].	
Step 12	[DDSApp01] offer the service [DDSService02].	
Step 13	[DDSApp03] request for the service [DDSService02].	Service discovery callback with a handle for service [DDSService02] should be received by [DDSApp03].
Step 14	[DDSApp03] register the EventReceiveHandler for Event_C.	
Step 15	[DDSApp03] subscribe for the service [DDSService02].	

▽

△

Step 16	[DDSApp01] send the Event_C.	Registered EventReceiveHandler should get triggered for Event_C in [DDSApp03] with data sent by [DDSApp01].
Step 17	[DDSApp01] stop offering the service [DDSService02].	

4.4.3 [STS_DDS_00003] Field communication using DDS binding.

Test Objective	To verify the Field communication using DDS binding.		
ID	STS_DDS_00003	**State**	Draft
Affected Functional Cluster	Communication Management		
Trace to RS Criteria	[RS_CM_00216], [RS_CM_00217], [RS_CM_00218], [RS_CM_00219], [RS_CM_00220], [RS_CM_00221], [RS_CM_00226], [RS_CM_00227]		
Reference to Test Environment	STC_CM_00001 in Test configurations Communication Management		
Configuration Parameters	- [DDSService03] is offered by [DDSApp01], requested by [DDSApp02] and [DDSApp03]. - Service: • DDSService03: – Field_A - Notifier, Setter and Getter - Deployment: • DdsServiceInterfaceDeployment. - Instance: • DdsProvidedServiceInstance • DdsRequiredServiceInstance Note: The service and Field names are arbitrary.		
Summary	[DDSApp01] and [DDSApp02] are on [ECU1] and [DDSApp03] is on [ECU2]. [DDSApp01], [DDSApp02] and [DDSApp03] are started when the machine state for [ECU1] and [ECU2] changes to Driving. [DDSApp01] offers the service [DDSService03]. [DDSApp02] and [DDSApp03] requests for the service [DDSService03]. [DDSApp02] subscribes for the service [DDSService03]. [DDSApp02] gets the value of Field_A and receives the initial value over service [DDSService03]. [DDSApp02] sets the value of Field_A over service [DDSService03]. [DDSApp01] updates the Field_A value. [DDSApp02] receives the notification with the updated field value. [DDSApp03] gets the value of Field_A and receives the value sent in set call of Field_A. [DDSApp03] subscribe for [DDSService03]. [DDSApp01] updates the Field_A value. [DDSApp03] sets the value of Field_A over the service [DDSService03]. Note: All the steps will be triggered by DDSTester and the result will be sent back to it.		
Pre-conditions	- [DDSTester] is connected to both ECU. - Both the ECUs are in machine state Parking.		
Post-conditions	[DDSTester] is disconnected from both ECUs.		

▽

△

Main Test Execution		
Test Steps		**Pass Criteria**
Step 1	[DDSTester] request the machine state change to Driving on [ECU1] and [ECU2].	Machine state on [ECU1] and [ECU2] changed to Driving.
Step 2	[DDSApp01] register the GetHandler and SetHandler for Field_A.	
Step 3	[DDSApp01] initialize the Field_A and offer the service [DDSService03].	
Step 4	[DDSApp02] request for the service [DDSService03].	Service discovery callback with a handle for service [DDSService03] should be received by [DDSApp02].
Step 5	[DDSApp03] request for the service [DDSService03].	Service discovery callback with a handle for service [DDSService03] should be received by [DDSApp03].
Step 6	[DDSApp02] subscribe for the service [DDSService03].	
Step 7	[DDSApp02] get the value of Field_A over the service [DDSService03].	[DDSApp02] should receive the initial value of Field_A.
Step 8	[DDSApp02] set the value of Field_A over the service [DDSService03].	[DDSApp02] should receive the return value with the data sent in set call of Field_A.
Step 9	[DDSApp01] update the value of Field_A.	[DDSApp02] should receive the notification with updated value of Field_A.
Step 10	[DDSApp03] get the value of Field_A over the service [DDSService03].	[DDSApp03] should receive the updated value of Field_A.
Step 11	[DDSApp03] subscribe for the service [DDSService03].	
Step 12	[DDSApp01] update the value of Field_A.	[DDSApp02] and [DDSApp03] should receive the notification with updated value of Field_A.
Step 13	[DDSApp03] set the value of Field_A over service [DDSService03].	[DDSApp03] should receive the return value and [DDSApp02] should receive the notification with the data sent in Set call of Field_A.

4.4.4 [STS_DDS_00004] Method communication using DDS binding.

Test Objective	To verify the Method communication using DDS binding.		
ID	STS_DDS_00004	**State**	Draft
Affected Functional Cluster	Communication Management		
Trace to RS Criteria	[RS_CM_00211], [RS_CM_00212], [RS_CM_00213], [RS_CM_00225], [RS_CM_00214], [RS_CM_00215]		
Reference to Test Environment	STC_CM_00001 in Test configurations Communication Management		
Configuration Parameters	- [DDSService04] is offered by [DDSApp01], requested by [DDSApp02]. - [DDSService05] is offered by [DDSApp01], requested by [DDSApp03]. - Service: • DDSService04: ▽		

▽

△

	△ – Method_A – Method_B • DDSService05: – Method_C – Method_D - fire & forget - Deployment: • DdsServiceInterfaceDeployment. - Instance: • DdsProvidedServiceInstance • DdsRequiredServiceInstance Note: The service and Method names are arbitrary.	
Summary	[DDSApp01] and [DDSApp02] are on [ECU1] and [DDSApp03] is on [ECU2]. [DDSApp01], [DDSApp02] and [DDSApp03] are started when the machine state for [ECU1] and [ECU2] changes to Driving. [DDSApp01] offers the service [DDSService04] with MethodCallProcessingMode as kPoll and [DDSApp02] requests for the same. [DDSApp02] calls the Method_A and receives the data synchronously from [DDSApp01]. [DDSApp02] calls the Method_B and receives the data synchronously from [DDSApp01]. [DDSApp01] offers the service [DDSService05] with MethodCallProcessingMode as kEvent and [DDSApp03] requests for the same. [DDSApp03] calls the Method_C and receives the data asynchronously from [DDSApp01]. [DDSApp03] calls the Method_D. Note: All the steps will be triggered by DDSTester and the result will be sent back to it.	
Pre-conditions	- [DDSTester] is connected to both ECU. - Both the ECUs are in machine state Parking.	
Post-conditions	[DDSTester] is disconnected from both ECUs.	
Main Test Execution		
Test Steps		**Pass Criteria**
Step 1	[DDSTester] request the machine state change to Driving on [ECU1] and [ECU2].	Machine state on [ECU1] and [ECU2] changed to Driving.
Step 2	[DDSApp01] offer the service [DDSService04].	
Step 3	[DDSApp02] request for the service [DDSService04].	Service discovery callback with a handle for service [DDSService04] should be received by [DDSApp02].
Step 4	[DDSApp02] receive the data synchronously by calling the Method_A over the service [DDSService04].	[DDSApp02] should receive the return data from the Method_A over the service [DDSService04].
Step 5	[DDSApp02] receive the data synchronously by calling the Method_B over the service [DDSService04].	[DDSApp02] should receive the return data from the Method_B over the service [DDSService04].
Step 6	[DDSApp01] offer the service [DDSService05].	
Step 7	[DDSApp03] request for the service [DDSService05].	Service discovery callback with a handle for service [DDSService05] should be received by [DDSApp03].
Step 8	[DDSApp03] receive the data asynchronously by calling the Method_C over the service [DDSService05].	[DDSApp02] should receive the return data from the Method_C over the service [DDSService05].
Step 9	[DDSApp03] call the Method_D over the service [DDSService05].	[DDSApp01] Method_D should get invoked with input data.

▽

△

Step 10	[DDSApp01] stop offering the service [DDSService04] and [DDSService05].	

5 Test configuration and test steps for Execution Management

5.1 Test System

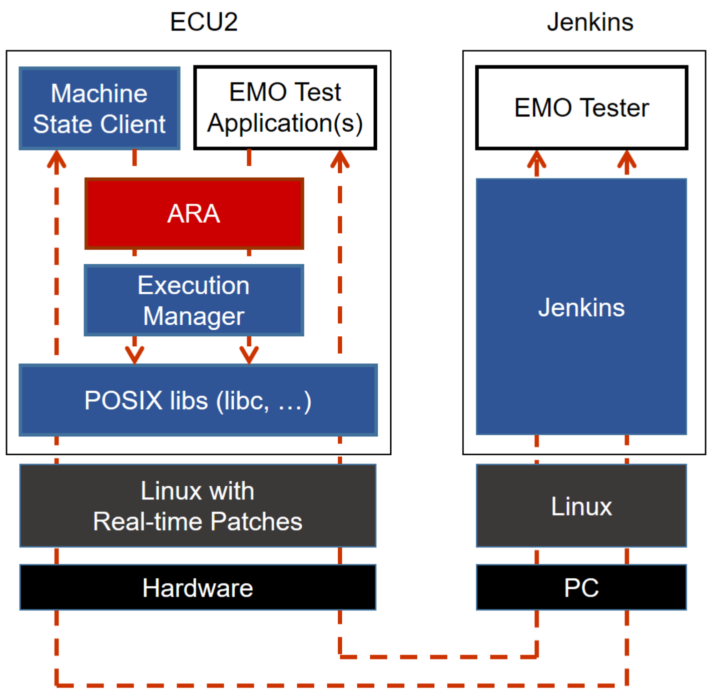

Figure 5.1: Illustration of test setup for Execution Management.

5.1.1 Test configurations

5.1.1.1 STC_EMO_00001

Configuration ID	STC_EMO_00001
Description	Standard Jenkins server for Execution Management test
ECU 2	Hardware, 192.168.100.2
Jenkins	Jenkins Server, 192.168.100.10

The Jenkins Server, running the job with the Execution Management test (Exec Tester) is connected via Ethernet to ECU2 hosting the System Test Applications [EMOApp02], [EMOApp03], [EMOApp04] and [EMOApp05].

The Exec Tester is supposed to check the pass criteria.

The communication between Exec Tester and the applications on the ECU may take place over the Diagnostics functional cluster in form of diagnostic messages.

5.1.1.1.1 Machine Manifest

Machine States	*Startup (Initial Mode)*
	Shutdown
	Restart
	Driving
	Parking

5.1.1.1.2 Application Manifest

Application Name	**EMOApp02**		
Process	ModeDependentStartupConfig	machineMode	*Driving*
Application Name	**EMOApp03**		
Process	ModeDependentStartupConfig	machineMode	*Driving*
Application Name	**EMOApp04**		
Process	ModeDependentStartupConfig	machineMode	*Driving*
Application Name	**EMOApp05**		
Process	ModeDependentStartupConfig	machineMode	*Driving*

5.1.1.2 STC_EMO_00002

Configuration ID	STC_EMO_00002
Description	Standard Jenkins server for Execution Management test
ECU 2	Hardware, 192.168.100.2
Jenkins	Jenkins Server, 192.168.100.10

The Jenkins Server, running the job with the Execution Management test (Exec Tester) is connected via Ethernet to ECU2 hosting the System Test Applications [EMOApp02], [EMOApp03], [EMOApp04], [EMOApp05] and [EMOApp06].

The Exec Tester is supposed to check the pass criteria.

The communication between Exec Tester and the applications on the ECU may take place over the Diagnostics functional cluster in form of diagnostic messages.

5.1.1.2.1 Machine Manifest

Machine States	*Startup (Initial Mode)*
	Shutdown
	Restart
	Driving
	Parking
Function Groups	
FG1	*Off*
	Running
	Fallback
	Diag
FG2	*Off*
	On
	Activate

5.1.1.2.2 Application Manifest

Application Name	**EMOApp02**		
Process	ModeDependentStartupConfig	machineMode	*Driving*
	DeterministicClient	cycleTimeValue	*CycleTimeValue1*
Application Name	**EMOApp03**		
Process	ModeDependentStartupConfig	machineMode	*Driving*
		executionDependency	[EMOApp02].*Running*
Application Name	**EMOApp04**		
Process	ModeDependentStartupConfig	machineMode	*Driving*
		executionDependency	[EMOApp03].*Running*
Application Name	**EMOApp05**		
Process	ModeDependentStartupConfig	functionGroup	[FG2].*On* and [FG2].*Activate*
Application Name	**EMOApp06**		
Process	ModeDependentStartupConfig	functionGroup	[FG2].*Activate*

5.1.1.3 STC_EMO_00003

Configuration ID	STC_EMO_00003
Description	Standard Jenkins server for Execution Management test
ECU 2	Hardware, 192.168.100.2
Jenkins	Jenkins Server, 192.168.100.10

The Jenkins Server, running the job with the Execution Management test (Exec Tester) is connected via Ethernet to ECU2 hosting the System Test Applications [EMOApp02], [EMOApp03], [EMOApp04] and [EMOApp05].

The Exec Tester is supposed to check the pass criteria.

The communication between Exec Tester and the applications on the ECU may take place over the Diagnostics functional cluster in form of diagnostic messages.

5.1.1.3.1 Machine Manifest

Machine States	*Startup (Initial Mode)*			
	Shutdown			
	Restart			
	Driving			
	Parking			
PerStateTimeout				
PerStateTimeout1	state	MachineState	*Driving*	
	timeout	EnterExit Timeout	enterTimeoutValue	*EnterTimeValue1*
			exitTimeoutValue	*ExitTimeValue1*
PerStateTimeout2	state	MachineState	*Parking*	
	timeout	EnterExit Timeout	enterTimeoutValue	*EnterTimeValue2*
			exitTimeoutValue	*ExitTimeValue2*

5.1.1.3.2 Application Manifest

Application Name	**EMOApp02**		
Process	ModeDependentStartupConfig	machineMode	*Driving*
Application Name	**EMOApp03**		
Process	ModeDependentStartupConfig	machineMode	*Driving*
Application Name	**EMOApp04**		
Process	ModeDependentStartupConfig	machineMode	*Parking*
Application Name	**EMOApp05**		
Process	ModeDependentStartupConfig	machineMode	*Parking*

5.1.1.3.3 ProcessToMachineMapping

Application Name	**EMOApp02**			
Process	shallRunOn	ProcessorCore	CoreId	1 and 2

▽

△

Application Name	**EMOApp03**			
Process	shallRunOn	ProcessorCore	CoreId	1 and 2
Application Name	**EMOApp04**			
Process	shallRunOn	ProcessorCore	CoreId	3 and 4
Application Name	**EMOApp05**			
Process	shallRunOn	ProcessorCore	CoreId	3 and 4

5.1.1.4 STC_EMO_00004

Configuration ID	STC_EMO_00004
Description	Standard Jenkins server for Execution Management test
ECU 2	Hardware, 192.168.100.2
Jenkins	Jenkins Server, 192.168.100.10

The Jenkins Server, running the job with the Execution Management test (Exec Tester) is connected via Ethernet to ECU2 hosting the System Test Applications [EMOApp02], [EMOApp03] and [EMOApp04].

The Exec Tester is supposed to check the pass criteria.

The communication between Exec Tester and the applications on the ECU may take place over the Diagnostics functional cluster in form of diagnostic messages.

5.1.1.4.1 Machine Manifest

Machine States	*Startup (Initial Mode)*
	Shutdown
	Restart
	Driving
	Parking
Function Groups	
FG1	*Off*
	On
	Activate
OsModuleInstantiation	
ResourceGroups	
ResourceGroup1	cpuUsage *CPULIM1*
	memUsage *MEMLIM1*
ResourceGroup2	cpuUsage *CPULIM2*

▽

△

	memUsage *MEMLIM2*

5.1.1.4.2 Application Manifest

Application Name	**EMOApp02**		
Process	ModeDependentStartupConfig	machineMode	*Driving*
		schedulingPolicy	*schedulingPolicyRoundRobin*
		schedulingPriority	3
Application Name	**EMOApp03**		
Process	ModeDependentStartupConfig	machineMode	*Driving*
		executionDependency	[EMOApp02].*Running*
		schedulingPolicy	*schedulingPolicyOther*
		schedulingPriority	0
Application Name	**EMOApp04**		
Process	ModeDependentStartupConfig	functionGroup	[FG1].*On*
		schedulingPolicy	*schedulingPolicyFifo*
		schedulingPriority	4
Application Name	**EMOApp05**		
Process1	ModeDependentStartupConfig	functionGroup	[FG1].*On*
		schedulingPolicy	*schedulingPolicyRoundRobin*
		schedulingPriority	1
		startupConfig	environmentVariable Key : APP_PATH Value : /home/user1
			startupOption optionArgument : inputfile_1 CommandLineOptionKindEnum : commandLineLongForm optionName : filename
Process2	ModeDependentStartupConfig	functionGroup	[FG2].*On*
		schedulingPolicy	*schedulingPolicyFifo*
		schedulingPriority	2
		startupConfig	environmentVariable Key : APP_PATH Value : /home/user2
			startupOption optionArgument : inputfile_2 CommandLineOptionKindEnum : commandLineLongForm optionName : filename

5.1.1.4.3 Process Configuration

Process Name	**Executable Reference**
EMOApp02Process	*EMOApp02Exec*
EMOApp03Process	*EMOApp03Exec*
EMOApp04Process	*EMOApp04Exec*
EMOApp05Process1	*EMOApp05Exec*
EMOApp05Process2	*EMOApp05Exec*

5.1.1.5 STC_EMO_00005

Configuration ID	STC_EMO_00005
Description	Standard Jenkins server for Execution Management test
ECU 2	Hardware, 192.168.100.2
Jenkins	Jenkins Server, 192.168.100.10

The Jenkins Server, running the job with the Execution Management test (Exec Tester) is connected via Ethernet to ECU2 hosting the System Test Applications [EMOApp02]

The Exec Tester is supposed to check the pass criteria.

The communication between Exec Tester and the applications on the ECU may take place over the Diagnostics functional cluster in form of diagnostic messages.

5.1.1.5.1 Application Manifest

Application Name	**EMOApp02**		
Process1	ModeDependentStartupConfig	functionGroup	[FG1].*On*
		cycleTimeValue	*TimeVal1*
		numberOfWorkers	2
Process2	ModeDependentStartupConfig	functionGroup	[FG2].*On*
		cycleTimeValue	*TimeVal1*
		numberOfWorkers	2

5.1.1.5.2 Process Configuration

Process Name	Executable Reference
EMOApp02Process1	*EMOApp02Exec*
EMOApp02Process2	*EMOApp02Exec*

5.2 Test cases

5.2.1 [STS_EMO_00001] Startup of applications with change of machine state.

Test Objective	Verification, that the execution management functional cluster can perform a change of Machine State and that applications associated with the new Machine State are started.		
ID	STS_EMO_00001	**State**	Draft
Affected Functional Cluster	Execution Management		
Trace to RS Criteria	[RS_EM_00100], [RS_EM_00101], [RS_EM_00103]		
Reference to Test Environment	STC_EMO_00001		
Configuration Parameters	• Machine State Driving, in which all System Test Applications [EMOApp02], [EMOApp03], [EMOApp04] and [EMOApp05] shall start is defined.		
Summary	When initialized the system state is *Startup*. A change of Machine State from *Startup* to *Parking* is requested and it is verified that [EMOApp02], [EMOApp03], [EMOApp04] and [EMOApp05] are not started. A change of Machine State from *Parking* to *Driving* is requested and the startup of the applications [EMOApp02], [EMOApp03], [EMOApp04] and [EMOApp05] associated with this Machine State is verified.		
Pre-conditions	- Exec Tester is connected to ECU2 via TCP. - Software components on ECU2 are initialized. - ECU2 is in Machine State *Startup*. - Operating system on ECU2 has booted.		
Post-conditions	TCP connection between Exec Tester and ECU2 is closed.		
Main Test Execution			
Test Steps			**Pass Criteria**
Step 1	[Exec Tester] Request change of Machine State to *Parking* for ECU2.		
Step 2	[SM] Request for change of Machine State to *Parking* from Execution Manager.		Machine State for ECU2 is changed to *Parking*.
Step 3	[Exec Tester] Query execution status of [EMOApp02].		[EMOApp02] is not executed.
Step 4	[Exec Tester] Query execution status of [EMOApp03].		[EMOApp03] is not executed.

▽

△

Step 5	[Exec Tester] Query execution status of [EMOApp04].	[EMOApp04] is not executed.
Step 6	[Exec Tester] Query execution status of [EMOApp05].	[EMOApp05] is not executed.
Step 7	[Exec Tester] Request change of Machine State to *Driving* for ECU2.	
Step 8	[SM] Request for change of Machine State to *Driving* from Execution Manager.	Machine State for ECU2 is changed to *Driving*.
Step 9	[Exec Tester] Query execution status of [EMOApp02].	[EMOApp02] is executed.
Step 10	[Exec Tester] Query execution status of [EMOApp03].	[EMOApp03] is executed.
Step 11	[Exec Tester] Query execution status of [EMOApp04].	[EMOApp04] is executed.
Step 12	[Exec Tester] Query execution status of [EMOApp05].	[EMOApp05] is executed.

5.2.2 [STS_EMO_00002] Shutdown of applications with change of machine state to Shutdown

Test Objective	Verification, that the execution management functional cluster executes a well-defined shutdown sequence for all configured and running applications, When shut-down is initiated		
ID	STS_EMO_00002	**State**	Draft
Affected Functional Cluster	Execution Management		
Trace to RS Criteria	[RS_EM_00100], [RS_EM_00101], [RS_EM_00103]		
Reference to Test Environment	STC_EMO_00001		
Configuration Parameters	- Machine State Driving, in which all System Test Applications [EMOApp02], [EMOApp03], [EMOApp04] and [EMOApp05] shall start is defined. - ECU ID for ECU2 is set to ECU2 - [EMOApp02] has LT Application ID APPID2. - Context ID for [EMOApp02] is set to CTX2 - [EMOApp03] has LT Application ID APPID3. - Context ID for [EMOApp03] is set to CTX3 - [EMOApp04] has LT Application ID APPID4. - Context ID for [EMOApp04] is set to CTX4 - [EMOApp05] has LT Application ID APPID5. - Context ID for [EMOApp05] is set to CTX5		
Summary	A change of Machine State from *Driving* to *Shutdown* is requested and the Shutdown of the applications [EMOApp02], [EMOApp03], [EMOApp04] and [EMOApp05] is verified by logging the messages at the termination of application.		

▽

△

Pre-conditions	- Exec Tester is connected to ECU2 via TCP. - Software components on ECU2 are initialized. - ECU2 is in Machine State Driving. - Operating system on ECU2 has booted. - Applications [EMOApp02], [EMOApp03], [EMOApp04] and [EMOApp05] are registered for logging and default log level is set to Verbose.	
Post-conditions	TCP connection between Exec Tester and ECU2 is closed.	
Main Test Execution		
Test Steps		**Pass Criteria**
Step 1	[Exec Tester] Request change of Machine State to *Shutdown* for ECU2.	
Step 2	[SM] Request for change of Machine State to *Shutdown* from Execution Manager.	Machine State for ECU2 is changed to *Shutdown*.
Step 3	[Exec Tester] Observe the log for applications [EMOApp02], [EMOApp03], [EMOApp04] and [EMOApp05]	Message with context ID CTX2 and application ID APPID2 is received which is logged at [EMOApp02] application termination Message with context ID CTX3 and application ID APPID3 is received which is logged at [EMOApp03] application termination Message with context ID CTX4 and application ID APPID4 is received which is logged at [EMOApp04] application termination Message with context ID CTX5 and application ID APPID5 is received which is logged at [EMOApp05] application termination

5.2.3 [STS_EMO_00003] Ordered Startup and Shutdown of Executables based on the dependency with other processes

Test Objective	Verification, that the execution management functional cluster can perform a change of Machine State and that applications associated with the new Machine State are started considering the dependency with other processes. Also to verify the ordered shutdown of the processes.		
ID	STS_EMO_00003	**State**	Draft
Affected Functional Cluster	Execution Management		
Trace to RS Criteria	[RS_EM_00100], [RS_EM_00101], [RS_EM_00103]		
Reference to Test Environment	STC_EMO_00002		

▽

△

Configuration Parameters	- Machine State *Driving*, in which System Test Applications [EMOApp02], [EMOApp03] and [EMOApp04] shall start is defined. Dependency with other process is configured as mentioned in section 5.2.1.2.2 Application Manifest. - ECU ID for ECU2 is set to *ECU2* - [EMOApp02] has LT Application ID *APPID2* - Context ID for [EMOApp02] is set to *CTX2* - [EMOApp03] has LT Application ID *APPID3* - Context ID for [EMOApp03] is set to *CTX3* - [EMOApp04] has LT Application ID *APPID4* - Context ID for [EMOApp04] is set to *CTX4* - [EMOApp05] has LT Application ID *APPID5* - Context ID for [EMOApp05] is set to *CTX5* - [EMOApp06] has LT Application ID *APPID6* - Context ID for [EMOApp06] is set to *CTX6*	
Summary	When initialized the system state is *Startup*. A change of Machine State from *Startup* to *Driving* is requested and the startup of the applications [EMOApp02], [EMOApp03] and [EMOApp04] associated with this Machine State are verified in the order of [EMOApp02], [EMOApp03] and [EMOApp04] by logging the messages at the Start of application processes. A change of Machine State from *Driving* to *Parking* is requested and the termination of the applications [EMOApp02], [EMOApp03] and [EMOApp04] is verified in the order of [EMOApp04], [EMOApp03] and [EMOApp02] by logging the messages at the termination of application processes.	
Pre-conditions	- Exec Tester is connected to ECU2 via TCP. - Software components on ECU2 are initialized. - ECU2 is in Machine State *Startup*. - Function Group State for [FG2] is *Off*. - Operating system on ECU2 has booted.	
Post-conditions	TCP connection between Exec Tester and ECU2 is closed.	
Main Test Execution		
Test Steps		**Pass Criteria**
Step 1	[Exec Tester] Request change of Machine State to *Driving* for ECU2.	
Step 2	[SM] Request for change of Machine State to *Driving* from Execution Manager.	Machine State for ECU2 is changed to *Driving*.
Step 3	[Exec Tester] Observe the log for applications [EMOApp02]	Message with context ID *CTX2* and application ID *APPID2* is received which is logged at [EMOApp02] application startup
Step 4	[Exec Tester] Observe the log for applications [EMOApp03]	Message with context ID *CTX3* and application ID *APPID3* is received which is logged at [EMOApp03] application startup
Step 5	[Exec Tester] Observe the log for applications [EMOApp04]	Message with context ID *CTX4* and application ID *APPID4* is received which is logged at [EMOApp04] application startup
Step 6	[Exec Tester] Request change of Machine State to *Shutdown* for ECU2.	

▽

△

Step 7	[SM] Request for change of Machine State to *Parking* from Execution Manager.	Machine State for ECU2 is changed to *Parking*.
Step 8	[Exec Tester] Observe the log for applications [EMOApp04]	Message with context ID *CTX4* and application ID *APPID4* is received which is logged at [EMOApp04] application termination
Step 9	[Exec Tester] Observe the log for applications [EMOApp03]	Message with context ID *CTX3* and application ID *APPID3* is received which is logged at [EMOApp03] application termination
Step 10	[Exec Tester] Observe the log for applications [EMOApp02]	Message with context ID *CTX2* and application ID *APPID2* is received which is logged at [EMOApp02] application termination

5.2.4 [STS_EMO_00004] Startup of applications with change of Function Group state

Test Objective	Verification, that the execution management functional cluster can perform a change of Function Group State and that Applications associated with the new Function Group State are started.		
ID	STS_EMO_00004	**State**	Draft
Affected Functional Cluster	Execution Management		
Trace to RS Criteria	[RS_EM_00100], [RS_EM_00101], [RS_EM_00103]		
Reference to Test Environment	STC_EMO_00002		
Configuration Parameters	- Function Group State *Activate* and Function Group State *On* of [FG2] in which System Test Application [EMOApp05] shall start is defined. - Function Group State *Activate* of [FG2] in which System Test Application [EMOApp06] shall start is defined		
Summary	When initialized the Function Group State of [FG2] is *Off*. A change of Function Group State of [FG2] to *On* is requested and the startup of the application [EMOApp05] associated with this Function Group State is verified. A change of Function Group State of [FG2] to *Activate* is requested and the startup of [EMOApp06] associated with this Function Group State is verified.		
Pre-conditions	- Exec Tester is connected to ECU2 via TCP. - Software components on ECU2 are initialized. - Function Group State [FG2] is *Off*. - Operating system on ECU2 has booted.		
Post-conditions	TCP connection between Exec Tester and ECU2 is closed.		
Main Test Execution			

▽

△

Test Steps		Pass Criteria
Step 1	[Exec Tester] Request change of Function Group State [FG2] to *On*.	
Step 2	[SM] Request for change of Function Group State [FG2] to *On* from Execution Manager.	Function Group State [FG2] for ECU2 is changed to *On*.
Step 3	[Exec Tester] Query execution status of [EMOApp05].	[EMOApp05] is executed.
Step 4	[Exec Tester] Request change of Function Group State [FG2] to *Activate*.	
Step 5	[SM] Request for change of Function Group State [FG2] to *Activate* from Execution Manager.	Function Group State [FG2] for ECU2 is changed to *Activate*.
Step 6	[Exec Tester] Query execution status of [EMOApp06].	[EMOApp06] is executed.

5.2.5 [STS_EMO_00005] Execution Management shall prevent Processes from directly starting other Processes

Test Objective	Verification that the execution management shall prevent Processes from directly starting other Processes		
ID	STS_EMO_00005	**State**	Draft
Affected Functional Cluster	Execution Management		
Trace to RS Criteria	[RS_EM_00009], [RS_EM_00100], [RS_EM_00101], [RS_EM_00103]		
Reference to Test Environment	STC_EMO_00003		
Configuration Parameters	- Machine State Driving, in which all System Test Applications [EMOApp02] and [EMOApp03] shall start is defined and Machine State Parking in which Applications [EMOApp04] and [EMOApp05] shall start is defined. - Each of the Applications [EMOApp02], [EMOApp03], [EMOApp04] and [EMOApp05] have one Executable invoked by a Process		
Summary	A change of Machine State from *Startup* to *Driving* is requested. Start of [EMOApp02] and [EMOApp03] Processes from Execution Manager is checked. Create or fork a Process from [EMOApp02] Process and verify that no child Processes are created from [EMOApp02] Process. Execute [EMOApp05] Process from [EMOApp03] Process and verify that the [EMOApp05] Process is not invoked from [EMOApp03] Process.		
Pre-conditions	- Exec Tester is connected to ECU2 via TCP. - Software components on ECU2 are initialized. - ECU2 is in Machine State *Startup*. - Operating system on ECU2 has booted.		

▽

△

Post-conditions	TCP connection between Exec Tester and ECU2 is closed.	
Main Test Execution		
Test Steps		**Pass Criteria**
Step 1	[Exec Tester] Request change of Machine State to Driving for ECU2.	
Step 2	[SM] Request for change of Machine State to *Driving* from Execution Manager.	Machine State for ECU2 is changed to *Driving*.
Step 3	Query execution status of [EMOApp02]	[EMOApp02] Process is executed
Step 4	[EMOApp02] Fork or create a Process from [EMOApp02]	
Step 5	[Exec Tester] Get the Process ID of the Execution Manager	Received the Process ID of Execution Manager. *EXMPID*
Step 6	[Exec Tester] Get the Process ID of [EMOApp02] Process	Received the Process ID of [EMOApp02] Process *APPID2*
Step 7	[Exec Tester] Get the Parent Process ID of [EMOApp02] Process	The Parent Process ID of [EMOApp02] Process is received as *EXMPID*
Step 8	[Exec Tester] Get the Child Processes of Process ID *APPID2*	No child Processes of [EMOApp02] Process shall be received.
Step 9	Query execution status of [EMOApp03]	[EMOApp03] Process is executed
Step 10	[EMOApp03] Execute or Invoke [EMOApp05] Process from [EMOApp03] Process	[EMOApp05] Process is not executed

5.2.6 [STS_EMO_00006] Execution Management shall create one POSIX process for each Executable instance and shall launch the process with the scheduling policy and priority configured in the Execution Manifest

Test Objective	Verification that the one POSIX process is created for each Executable instance configured and the scheduling policy and priority for the process is assigned as specified in the Execution Manifest.		
ID	STS_EMO_00006	**State**	Draft
Affected Functional Cluster	Execution Management		
Trace to RS Criteria	[RS_EM_00002], [RS_EM_00009], [RS_EM_00100], [RS_EM_00101], [RS_EM_00103]		
Reference to Test Environment	STC_EMO_00004		

▽

△

Configuration Parameters	- Machine State Driving, in which Processes [EMOApp02].Process and [EMOApp03].Process shall start is defined with [EMOApp03].Process having dependency on [EMOApp02].Process The scheduling policy and scheduling priority are configured as schedulingPolicyRoundRobin and 3 respectively for [EMOApp02].Process and schedulingPolicyOther and 0 respectively for [EMOApp03].Process - Function Group State On of [FG2] in which Process [EMOApp04].Process shall start is defined with scheduling policy as schedulingPolicyFifo and scheduling priority 4.	
Summary	A change of Machine State from Startup to Driving is requested. Start of [EMOApp02].Process from the Execution Manager with the configured scheduling policy (schedulingPolicyRoundRobin) and priority (3) is checked. Start of [EMOApp03].Process from the Execution Manager with the configured scheduling policy (schedulingPolicyOther) and priority (0) is checked after the start of [EMOApp02].Process, since [EMOApp03].Process has dependency on [EMOApp02].Process A change of Function Group State of [FG1] to On is requested and the startup of the Process [EMOApp04].Process is verified with the configured scheduling policy (schedulingPolicyFifo) and scheduling priority (4).	
Pre-conditions	- Exec Tester is connected to ECU2 via TCP. - Software components on ECU2 are initialized. - ECU2 is in Machine State *Startup*. - ECU2 Function Group State [FG2] is *Off*. - Operating system on ECU2 has booted.	
Post-conditions	TCP connection between Exec Tester and ECU2 is closed.	
Main Test Execution		
Test Steps		**Pass Criteria**
Step 1	[Exec Tester] Request change of Machine State to Driving for ECU2.	
Step 2	[SM] Request for change of Machine State to *Driving* from Execution Manager.	Machine State for ECU2 is changed to *Driving*.
Step 3	[Exec Tester] Query execution status of [EMOApp02] Process	[EMOApp02] Process is executed
Step 4	[Exec Tester] Get the Process ID of the Execution Manager	Received the Process ID of Execution Manager. *EXMPID*
Step 5	[Exec Tester] Get the Process ID of the [EMOApp02] Process	Received the Process ID of [EMOApp02] Process. *APPID2*
Step 6	[Exec Tester] Get the Parent Process ID of [EMOApp02]	The Parent Process ID of [EMOApp02] is received as *EXMPID*
Step 7	[Exec Tester] Get the scheduling policy of [EMOApp02] Process	Scheduling policy is received as SCHED_RR
Step 8	[Exec Tester] Get the scheduling priority of [EMOApp02] Process	Scheduling priority is received as 3
Step 9	[Exec Tester] Get the Process ID of the [EMOApp03] Process	Received the Process ID of [EMOApp03] Process. *APPID3*
Step 10	[Exec Tester] Get the Parent Process ID of [EMOApp03]	The Parent Process ID of [EMOApp03] is received as *EXMPID*

▽

△

Step 11	[Exec Tester] Get the scheduling policy of [EMOApp03] Process	Scheduling policy is received as SCHED_OTHER
Step 12	[Exec Tester] Get the scheduling priority of [EMOApp02] Process	Scheduling priority is received as 0
Step 13	[SM] Request change of Function Group State [FG2] to *On*.	
Step 14	[Exec Tester] Request for change of Function Group State [FG2] to *On* from Execution Manager.	Function Group State [FG2] for ECU2 is changed to *On*.
Step 15	[Exec Tester] Get the Process ID of the [EMOApp04] Process	Received the Process ID of [EMOApp04] Process. *APPID4*
Step 16	[Exec Tester] Get the Parent Process ID of [EMOApp04]	The Parent Process ID of [EMOApp04] is received as *EXMPID*
Step 17	[Exec Tester] Get the scheduling policy of [EMOApp04] Process	Scheduling policy is received as SCHED_FIFO
Step 18	[Exec Tester] Get the scheduling priority of [EMOApp04] Process	Scheduling priority is received as 4

5.2.7 [STS_EMO_00007] Execution Management shall support multiple instantiation of Executable with different startup parameters from different Processes

Test Objective	Verification that Execution Management shall support multiple instantiation of Executable from different POSIX processes with different startup parameters.		
ID	STS_EMO_00007	**State**	Draft
Affected Functional Cluster	Execution Management		
Trace to RS Criteria	[RS_EM_00010], [RS_EM_00002], [RS_EM_00100], [RS_EM_00101], [RS_EM_00103]		
Reference to Test Environment	STC_EMO_00004		
Configuration Parameters	Function Group State *On* of [FG1] in which Process [EMOApp05].Process1 shall start is defined with following StartupConfig schedulingPolicy : schedulingPolicyRoundRobin schedulingPriority : 1 StartupOption : filename = inputfile_1 Environment Variable : APP_PATH = /home/user1 Function Group State *On* of [FG1] in which Process [EMOApp05].Process2 shall start is defined with following StartupConfig ▽		

▽

△

	△ schedulingPolicy : schedulingPolicyFifo schedulingPriority : 2 StartupOption : filename = inputfile_2 Environment Variable : APP_PATH = /home/user2	
Summary	A change of Function Group State of [FG1] to *On* is requested. startup of the Process [EMOApp05].Process1 is verified A change of Function Group State of [FG2] to *On* is requested. startup of the Process [EMOApp05].Process2 is verified It is verified that the same Executable *EMOApp05Exec* is invoked from both the Processes [EMOApp05].Process1 and [EMOApp05].Process2 with different startup parameters as specified below: [EMOApp05].Process1 scheduling policy : schedulingPolicyRoundRobin scheduling priority : 1 argument : filename = inputfile_1 environment variable : APP_PATH = /home/user1 [EMOApp05].Process2 scheduling policy : schedulingPolicyFifo scheduling priority : 2 argument : filename = inputfile_2 environment variable : APP_PATH = /home/user2 Note: *EMOApp05Exec* shall invoke a main program with 3 arguments which specifies argument count, argument list and environment list.	
Pre-conditions	- Exec Tester is connected to ECU2 via TCP. - Software components on ECU2 are initialized. - ECU2 is in Machine State *Startup*. - ECU2 Function Group State [FG2] is *Off*. - Operating system on ECU2 has booted.	
Post-conditions	TCP connection between Exec Tester and ECU2 is closed.	
Main Test Execution		
Test Steps		**Pass Criteria**
Step 1	[Exec Tester] Request change of Function Group State [FG1] to *On*.	
Step 2	[SM] Request for change of Function Group State [FG1] to *On* from Execution Manager	Function Group State [FG1] for ECU2 is changed to *On*.
Step 3	[Exec Tester] Query execution status of [EMOApp05].Process1	[EMOApp05].Process1 is executed
Step 4	[Exec Tester] Get the Process ID of the [EMOApp05].Process1	Received the Process ID of [EMOApp05].Process1 *APPID5*
Step 5	[Exec Tester] Get the scheduling policy of [EMOApp05].Process1	Scheduling policy is received as SCHED_RR
Step 6	[Exec Tester] Get the scheduling priority of [EMOApp05].Process1	Scheduling priority is received as 1

▽

△

Step 7	[EMOApp05].Process1 Read the arguments	Scheduling policy is received as SCHED_RR
Step 8	[Exec Tester] Get the arguments of [EMOApp05].Process1	Check if only one argument is received and the argument received is filename = inputfile_1
Step 9	[EMOApp05].Process1 Read the environment variables	
Step 10	[Exec Tester] Get the environment variables of [EMOApp05].Process1	Check if the environment variable APP_PATH has /home/user1
Step 11	[Exec Tester] Request change of Function Group State [FG2] to *On*.	
Step 12	[SM] Request for change of Function Group State [FG2] to *On* from Execution Manager	Function Group State [FG2] for ECU2 is changed to *On*.
Step 13	[Exec Tester] Query execution status of [EMOApp05].Process2	[EMOApp05].Process2 is executed
Step 14	[Exec Tester] Get the Process ID of the [EMOApp05].Process2	Received the Process ID of [EMOApp05].Process2 *APPID5*
Step 15	[Exec Tester] Get the scheduling policy of [EMOApp05].Process2	Scheduling policy is received as SCHED_FIFO
Step 16	[Exec Tester] Get the scheduling priority of [EMOApp05].Process2	Scheduling priority is received as 2
Step 17	[EMOApp05].Process2 Read the arguments	
Step 18	[Exec Tester] Get the arguments of [EMOApp05].Process2	Check if only one argument is received and the argument received is filename = inputfile_2
Step 19	[EMOApp05].Process1 Read the environment variables	
Step 20	[Exec Tester] Get the environment variables of [EMOApp05].Process2	Check if the environment variable APP_PATH has /home/user2

5.2.8 [STS_EMO_00008] Execution Management shall support self initiated graceful shutdown of Processes

Test Objective	Verification that Execution Management shall support self initiated graceful shutdown of processes.		
ID	STS_EMO_00008	**State**	Draft

▽

△

Affected Functional Cluster	Execution Management	
Trace to RS Criteria	[RS_EM_00011], [RS_EM_00100], [RS_EM_00101], [RS_EM_00103]	
Reference to Test Environment	STC_EMO_00003	
Configuration Parameters	Machine State Driving, in which all System Test Applications [EMOApp02] shall start is defined	
Summary	A change of Machine State from Startup to Driving is requested. Start of [EMOApp02] Process is checked. Initiate self termination from [EMOApp02] Process and check that Execution Manager supports the self initiated shutdown of Process	
Pre-conditions	- Exec Tester is connected to ECU2 via TCP. - Software components on ECU2 are initialized. - ECU2 is in Machine State *Startup*. - Operating system on ECU2 has booted.	
Post-conditions	TCP connection between Exec Tester and ECU2 is closed.	
Main Test Execution		
Test Steps		**Pass Criteria**
Step 1	[Exec Tester] Request change of Machine State to *Driving* for ECU2.	
Step 2	[SM] Request for change of Machine State to *Driving* from Execution Manager.	Machine State for ECU2 is changed to *Driving*.
Step 3	[Exec Tester] Query execution status of [EMOApp02] Process	[EMOApp02] Process is executed
Step 4	[Exec Tester] Get the Process ID of the [EMOApp02] Process1	Received the Process ID of [EMOApp02] Process *APPID2*
Step 5	[EMOApp02] Process Report kTerminating state using API ExecutionClient::ReportExecutionState to Execution Manager	
Step 6	[EMOApp02] Process Exit from [EMOApp02] Process	
Step 7	[Exec Tester] Get the list of currently running process	Check if *APPID2* does not exist in the list of currently running process

5.2.9 [STS_EMO_00009] Execution Management shall support binding of processes and its associated threads to specified set of cores

Test Objective	Verification that the Execution Management shall support the binding of processes and its associated threads to specific set of cores as specified in the Execution Manifest.		
ID	STS_EMO_00009	**State**	Draft
Affected Functional Cluster	Execution Management		
Trace to RS Criteria	[RS_EM_00008], [RS_EM_00100], [RS_EM_00101], [RS_EM_00103]		
Reference to Test Environment	STC_EMO_00003		
Configuration Parameters	- Machine State Driving, in which all System Test Applications [EMOApp02], [EMOApp03], [EMOApp04] and [EMOApp05] shall start is defined - [EMOApp02].Process and [EMOApp03].Process are mapped to cores 1 and 2 - [EMOApp04].Process and [EMOApp05].Process are mapped to cores 3 and 4		
Summary	A change of Machine State from Startup to Driving is requested. Start of [EMOApp02] Process is checked. Also it is checked that [EMOApp02] Process runs on core 1 and 2 as configured in the Execution Manifest. Threads are created inside the [EMOApp02] Process and it is checked that threads are running on core 1 or 2. Assign core 1 to thread created inside [EMOApp02] Process and it is checked that the thread runs in core 1. Assign core 3 to thread created inside [EMOApp02] Process and it is checked that the thread does not run in core 3, since core 3 is not set for [EMOApp02] Process		
Pre-conditions	- Exec Tester is connected to ECU2 via TCP. - Software components on ECU2 are initialized. - ECU2 is in Machine State *Startup*. - Operating system on ECU2 has booted.		
Post-conditions	TCP connection between Exec Tester and ECU2 is closed.		
Main Test Execution			
Test Steps			**Pass Criteria**
Step 1	[Exec Tester] Request change of Machine State to *Driving* for ECU2.		
Step 2	[SM] Request for change of Machine State to *Driving* from Execution Manager.		Machine State for ECU2 is changed to *Driving*.
Step 3	[Exec Tester] Query execution status of [EMOApp02] Process		[EMOApp02] Process is executed
Step 4	[Exec Tester] Get the Process ID of the [EMOApp02] Process1		Received the Process ID of [EMOApp02] Process *APPID2*
Step 5	[Exec Tester] Get the core in which [EMOApp02] Process is running		Check if the [EMOApp02] Process is running in core 1 or 2
Step 6	[EMOApp02] Process Create a thread *APP2ProcThread1* inside the [EMOApp02] Process		

▽

△

Step 7	[Exec Tester] Get the core in which the thread *APP2ProcThread1* is running	Check if the thread *APP2ProcThread1* is running in core 1 or 2
Step 8	[EMOApp02] Process Assign core 1 to the thread *APP2ProcThread1*	
Step 9	[Exec Tester] Get the core in which the thread *APP2ProcThread1* is running	Check if the thread *APP2ProcThread1* is running in core 1
Step 10	[EMOApp02] Process Create a thread *APP2ProcThread2* inside the [EMOApp02] Process	
Step 11	[Exec Tester] Get the core in which the thread *APP2ProcThread2* is running	Check if the thread *APP2ProcThread2* is running in core 1 or 2
Step 12	[EMOApp02] Process Assign core 3 to the thread *APP2ProcThread2*	
Step 13	[Exec Tester] Get the core in which the thread *APP2ProcThread2* is running	Check if the thread *APP2ProcThread2* is running in core 1 or 2

5.2.10 [STS_EMO_00010] Execution Management shall support the configuration of OS resource budgets for Process and group of Processes

Test Objective	Verification that the execution management shall assign the ResourceGroup to process or group of processes based on the configuration in the Execution Manifest and also to verify that the CPU limit and memory limit assigned to ResourceGroup is based on the configuration in the Execution Manifest.		
ID	STS_EMO_00010	**State**	Draft
Affected Functional Cluster	Execution Management		
Trace to RS Criteria	[RS_EM_00005], [RS_EM_00100], [RS_EM_00101], [RS_EM_00103]		
Reference to Test Environment	STC_EMO_00004		
Configuration Parameters	- Machine State Driving, in which System Test Applications [EMOApp02] and [EMOApp03] shall start is defined - Function Group State On of [FG1] in which [EMOApp04] Process1 shall start is defined - Two ResourceGroups *ResourceGroup1* and *ResourceGroup2* are configured - *ResourceGroup1* is configured with CPU limit and Memory limit as *CPULIM1* and *MEMLIM1* respectively. *ResourceGroup2* is configured with CPU limit and Memory limit as *CPULIM2* and *MEMLIM2* respectively - [EMOApp02] and [EMOApp03] Process are mapped to *ResourceGroup1* and [EMOApp04] Process is mapped to *ResourceGroup2* Hint: CPU limit is specified as percentage of the total CPU capacity on the machine and Memory limit is specified in bytes		

▽

△

Summary	A change of Machine State from *Startup* to *Driving* is requested. Start of [EMOApp02] Process is checked. Then start of [EMOApp03] Process is checked Get the Resource Group of [EMOApp02] and [EMOApp03] Process and check if the Resource Group assigned is *ResourceGroup1* Get the CPU and Memory limit of Resource Group *ResourceGroup1* and check if the CPU limit and Memory limit are *CPULIM1* and *MEMLIM1* respectively. A change of Function Group State of [FG1] to On is requested and startup of the [EMOApp04] Process is verified Get the Resource Group of [EMOApp04] Process and check if the Resource Group assigned is *ResourceGroup2*. Get the CPU and Memory limit of Resource Group *ResourceGroup2* and check if the CPU limit and Memory limit are *CPULIM2* and *MEMLIM2* respectively.	
Pre-conditions	- Exec Tester is connected to ECU2 via TCP. - Software components on ECU2 are initialized. - ECU2 is in Machine State *Startup*. - ECU2 Function Group State [FG1] is *Off* - Operating system on ECU2 has booted.	
Post-conditions	TCP connection between Exec Tester and ECU2 is closed.	
Main Test Execution		
Test Steps		**Pass Criteria**
Step 1	[Exec Tester] Request change of Machine State to *Driving* for ECU2.	
Step 2	[SM] Request for change of Machine State to *Driving* from Execution Manager.	Machine State for ECU2 is changed to *Driving*.
Step 3	[Exec Tester] Query execution status of [EMOApp02] Process	[EMOApp02] Process is executed
Step 4	[Exec Tester] Get the ResourceGroup of [EMOApp02] Process	ResourceGroup is received as *ResourceGroup1*
Step 5	[Exec Tester] Get the CPU limit of *ResourceGroup1*	CPU limit is received as *CPULIM1*
Step 6	[Exec Tester] Get the Memory limit of *ResourceGroup1*	Memory limit is received as *MEMLIM1*
Step 7	[Exec Tester] Query execution status of [EMOApp03]	[EMOApp03] Process is executed
Step 8	[Exec Tester] Get the ResourceGroup of [EMOApp03] Process	ResourceGroup is received as *ResourceGroup1*
Step 9	[Exec Tester] Request change of Function Group State [FG1] to *On*	
Step 10	[SM] Request for change of Function Group State [FG1] to On from Execution Manager.	Function Group State [FG1] for ECU2 is changed to *On*.
Step 11	[Exec Tester] Query execution status of [EMOApp04] Process	[EMOApp04] Process is executed
Step 12	[Exec Tester] Get the ResourceGroup of [EMOApp04] Process	ResourceGroup is received as *ResourceGroup2*
Step 13	[Exec Tester] Get the CPU limit of *ResourceGroup2*	CPU limit is received as *CPULIM2*
Step 14	[Exec Tester] Get the Memory limit of *ResourceGroup2*	Memory limit is received as *MEMLIM2*

5.2.11 [STS_EMO_00011] Execution Management shall support recovery actions in case an Process deviates from normal behavior

Test Objective	Verification that the Execution Manager shall support recovery actions when the Process is not terminated within the configured exit timeout value.		
ID	STS_EMO_00011	**State**	Draft
Affected Functional Cluster	Execution Management		
Trace to RS Criteria	[RS_EM_00013], [RS_EM_00100], [RS_EM_00101], [RS_EM_00103]		
Reference to Test Environment	STC_EMO_00003		
Configuration Parameters	- Machine States *Driving* and *Parking* are configured - Machine State *Driving*, in which System Test Applications [EMOApp02] and [EMOApp03] shall start is defined - exitTimeoutValue is configured as *ExitTimeVal1* for Machine State Driving		
Summary	A change of Machine State from *Startup* to *Driving* is requested. Start of [EMOApp02] and [EMOApp03] Process is checked A change of Machine State from *Driving* to *Parking* is requested. [EMOApp02] Process is not terminated within the configured exitTimeoutValue *ExitTimeVal1* Execution Manager notifies Platform Health Management that timeout is detected for [EMOApp02] Process. Platform Health Management shall trigger Recovery action to restart the Process.		
Pre-conditions	- Exec Tester is connected to ECU2 via TCP. - Software components on ECU2 are initialized. - ECU2 is in Machine State *Startup*. - Operating system on ECU2 has booted.		
Post-conditions	TCP connection between Exec Tester and ECU2 is closed.		
Main Test Execution			
Test Steps			**Pass Criteria**
Step 1	[Exec Tester] Query execution status of [PHM].		[PHM] is started
Step 2	[Exec Tester] Request change of Machine State to *Driving* for ECU2.		
Step 3	[SM] Request for change of Machine State to *Driving* from Execution Manager.		Machine State for ECU2 is changed to *Driving*.
Step 4	[Exec Tester] Query execution status of [EMOApp02] Process		[EMOApp02] Process is executed
Step 5	[Exec Tester] Query execution status of [EMOApp03] Process		[EMOApp03] Process is executed
Step 6	[Exec Tester] Request change of Machine State to *Parking* for ECU2.		
Step 7	[SM] Request for change of Machine State to *Parking* from Execution Manager.		Machine State for ECU2 is changed to *Parking*.

▽

△

Step 8	[Exec Tester] Start *ExitTimeVal1* timer	[EMOApp02] Process is executed
Step 9	[Exec Tester] After the *ExitTimeVal1* timer expires. Query execution status of [EMOApp02] Process	[EMOApp02] Process is not terminated.
Step 10	[EXM] Execution Manager shall notify Platform Health Management about timeout	
Step 11	[PHM] Request to Execution Manager to Restart the [EMOApp02] Process	Operation succeeded
Step 12	[EXM] Report error to State Manager that the state transition request is not fulfilled	State change request could not be finished in time

5.2.12 [STS_EMO_00012] Only Execution Management shall start Processes

Test Objective	Verification that all the processes are started by Execution Manager other than system specific processes directly started by the OS outside of AP.		
ID	STS_EMO_00012	**State**	Draft
Affected Functional Cluster	Execution Management		
Trace to RS Criteria	[RS_EM_00009], [RS_EM_00100], [RS_EM_00101], [RS_EM_00103],		
Reference to Test Environment	STC_EMO_00003		
Configuration Parameters	- Machine State Driving, in which System Test Applications [EMOApp02] and [EMOApp03] shall start is defined - Machine State Parking, in which System Test Applications [EMOApp04] and [EMOApp05] shall start is defined		
Summary	A change of Machine State from *Startup* to *Driving* is requested. Start of [EMOApp02] and [EMOApp03] Process is checked Get the parent Process ID of [EMOApp02] and [EMOApp03] Process and check if it is equal to the Process Id of Execution Manager A change of Machine State from Driving to Parking is requested. Start of [EMOApp04] and [EMOApp05] Process is checked Get the parent Process ID of [EMOApp04] and [EMOApp05] Process and check if it is equal to the Process Id of Execution Manager Check if all the Application Processes which are configred in the Application Manifest files are invoked by Execution Manager		
Pre-conditions	- Exec Tester is connected to ECU2 via TCP. - Software components on ECU2 are initialized. - ECU2 is in Machine State *Startup*. - Operating system on ECU2 has booted.		
Post-conditions	TCP connection between Exec Tester and ECU2 is closed.		
Main Test Execution			

▽

△

Test Steps		Pass Criteria
Step 1	[Exec Tester] Request change of Machine State to *Driving* for ECU2.	
Step 2	[SM] Request for change of Machine State to *Driving* from Execution Manager.	Machine State for ECU2 is changed to *Driving*.
Step 3	[Exec Tester] Get the Process ID of the Execution Manager	Received the Process ID of Execution Manager. *EXMPID*
Step 4	[Exec Tester] Query execution status of [EMOApp02] Process	[EMOApp02] Process is executed
Step 5	[Exec Tester] Query execution status of [EMOApp03] Process	[EMOApp03] Process is executed
Step 6	[Exec Tester] Get the Process ID of [EMOApp02] Process	Received the Process ID of [EMOApp02] Process *APPID2*
Step 7	[Exec Tester] Get the Parent Process ID of [EMOApp02] Process	The Parent Process ID of [EMOApp02] Process is received as *EXMPID*
Step 8	[Exec Tester] Get the Process ID of [EMOApp03] Process	Received the Process ID of [EMOApp03] Process *APPID3*
Step 9	[Exec Tester] Get the Parent Process ID of [EMOApp03] Process	The Parent Process ID of [EMOApp03] Process is received as *EXMPID*
Step 10	[Exec Tester] Request change of Machine State to *Parking* for ECU2.	
Step 11	[SM] Request for change of Machine State to *Parking* from Execution Manager.	Machine State for ECU2 is changed to *Parking*.
Step 12	[Exec Tester] Query execution status of [EMOApp04] Process	[EMOApp04] Process is executed
Step 13	[Exec Tester] Query execution status of [EMOApp05] Process	[EMOApp05] Process is executed
Step 14	[Exec Tester] Get the Process ID of [EMOApp04] Process	Received the Process ID of [EMOApp04] Process *APPID4*
Step 15	[Exec Tester] Get the Parent Process ID of [EMOApp04] Process	The Parent Process ID of [EMOApp04] Process is received as *EXMPID*
Step 16	[Exec Tester] Get the Process ID of [EMOApp05] Process	Received the Process ID of [EMOApp05] Process *APPID5*
Step 17	[Exec Tester] Get the Parent Process ID of [EMOApp05] Process	The Parent Process ID of [EMOApp05] Process is received as *EXMPID*

5.2.13 [STS_EMO_00013] The API provided by Execution Management shall be used by the Processes for cyclic triggering of its activities

Test Objective	Verification of the API provided by Execution Management for cyclic triggering of the activities of Processes.		
ID	STS_EMO_00012	**State**	Draft
Affected Functional Cluster	Execution Management		
Trace to RS Criteria	[RS_EM_00052], [RS_EM_00100], [RS_EM_00101], [RS_EM_00103]		
Reference to Test Environment	STC_EMO_00002		
Configuration Parameters	- Machine State Driving, in which System Test Application [EMOApp02] shall start is defined - cycleTimeValue for [EMOApp02] Process is configured as *CycleTimeValue1* - [EMOApp02] has LT Application ID APPID2 - Context ID for [EMOApp02] is set to CTX2		
Summary	A change of Machine State from *Startup* to *Driving* is requested. Start of [EMOApp02] Process is checked. Create a continuous loop in [EMOApp02] Process. Inside the loop invoke WaitForNextActivation and then invoke DLT log message to log the current time stamp value, if return value of WaitForNextActivation is kRun. Check if the difference in time stamp value between two DLT log messages is *CycleTimeValue1*.		
Pre-conditions	- Exec Tester is connected to ECU2 via TCP. - Software components on ECU2 are initialized. - ECU2 is in Machine State *Startup*. - Operating system on ECU2 has booted.		
Post-conditions	TCP connection between Exec Tester and ECU2 is closed.		
Main Test Execution			
Test Steps			**Pass Criteria**
Step 1	[Exec Tester] Request change of Machine State to *Driving* for ECU2.		
Step 2	[SM] Request for change of Machine State to *Driving* from Execution Manager.		Machine State for ECU2 is changed to *Driving*.
Step 3	[Exec Tester] Query execution status of [EMOApp02] Process		[EMOApp02] Process is executed
Step 4	[EMOApp02] Check if the Process reports ExecutionState kRunning to Execution Manager		ExecutionState kRunning is reported
Step 5	[EMOApp02] Start a Continuous While Loop		
Step 6	[EMOApp02] Invoke DeterministicClient::WaitForNextActivation Hint: The call shall be made inside while loop		Check if Deterministic-Client::WaitForNextActivation returns kRun

▽

△

Step 7	[EMOApp02] Invoke DLT log message to log the current time stamp value Hint: The call shall be made inside while loop after WaitForNextActivation	
Step 8	[Exec Tester] Observe the log messages of [EMOApp02] Process	Check if the difference in time stamp value between two consecutive DLT messages is *CycleTimeValue1*

5.2.14 [STS_EMO_00014] Execution Management shall provide API to the Process to support deterministic redundant execution of the process

Test Objective	Verification that the Execution Management shall provide API to the Processes to support deterministic redundant execution of the process.		
ID	STS_EMO_00014	**State**	Draft
Affected Functional Cluster	Execution Management		
Trace to RS Criteria	[RS_EM_00052], [RS_EM_00053], [RS_EM_00100], [RS_EM_00101], [RS_EM_00103]		
Reference to Test Environment	STC_EMO_00005		
Configuration Parameters	- Function Group State On of [FG1] in which Process [EMOApp02].Process1 and Process [EMOApp02].Process2 shall start is defined - Same EMOApp02Exec is invoked from both [EMOApp02].Process1 and [EMOApp02].Process2		
Summary	Create a continuous loop in EMOAPP2Exec which is triggered by [EMOApp02].Process1 and [EMOApp02].Process2 Invoke DeterministicClient::WaitForNextActivation API inside the loop. Check if the return value kRegisterServices, kServiceDiscovery, kInit, kRun is received in a sequence by both the Processes [EMOApp02].Process1 and [EMOApp02].Process2 Once the return value of WaitForNextActivation is kRun. Invoke DeterministicClient::RunWorkerPool API with 2 container elements. Invoke DeterministicClient::GetRandom() API for container element <ContElem1> and check if the returned random number is <RandNum1> for [EMOApp02].Process1 and also check if the returned random number is <RandNum1> for [EMOApp02].Process2 Invoke DeterministicClient::GetRandom() API for container element <ContElem2> and check if the returned random number is <RandNum2> for [EMOApp02].Process1 and also check if the returned random number is <RandNum2> for [EMOApp02].Process2 Invoke DeterministicClient::GetActivationTime API and check if the time value is <TimeVal1_1> for [EMOApp02].Process1 and also check for [EMOApp02].Process2 if the time value is <TimeVal1_1> Invoke DeterministicClient::GetNextActivationTime and check if the time value is <TimeVal1_2> for [EMOApp02].Process1 and also check if the time value is <TimeVal1_2> for [EMOApp02].Process2 Hint: The API's RunWorkerPool, GetRandom, GetActivationTime, GetNextActivationTime shall be invoked within the loop, only when the return value of WaitForNextActivation is kRun. Also the API's shall be invoked within the cycle time value <TimeVal1>, before WaitForNextActivation is triggered again.		
Pre-conditions	- Exec Tester is connected to ECU2 via TCP. - Software components on ECU2 are initialized. - ECU2 is in Machine State *Startup*. - Operating system on ECU2 has booted.		

▽

△

Post-conditions	TCP connection between Exec Tester and ECU2 is closed.	
Main Test Execution		
Test Steps		**Pass Criteria**
Step 1	[Exec Tester] Request change of Function Group State [FG1] to *On*.	
Step 2	[SM] Request for change of Function Group State [FG1] to *On* from Execution Manager.	Function Group State [FG1] for ECU2 is changed to *On*.
Step 3	[Exec Tester] Query execution status of [EMOApp02] Process1	[EMOApp02] Process1 is executed
Step 4	[Exec Tester] Query execution status of [EMOApp02] Process2	[EMOApp02] Process2 is executed
Step 5	[EMOApp02] Check if the Process reports ExecutionState kRunning to Execution Manager	ExecutionState kRunning is reported by [EMOApp02] Process1 ExecutionState kRunning is reported by [EMOApp02] Process2
Step 6	[EMOApp02] Start a Continuous While Loop	
Step 7	[EMOApp02] Invoke DeterministicClient::WaitForNextActivation. Hint: The call shall be made inside while loop	Check if the return value of Deterministic-Client::WaitForNextActivation is kRegisterServices for [EMOApp02].Process1 Check if the return value of Deterministic-Client::WaitForNextActivation is kRegisterServices for [EMOApp02].Process2
Step 8	[EMOApp02] Observe the return value of DeterministicClient::WaitForNextActivation.	Check if the return value of Deterministic-Client::WaitForNextActivation is kServiceDiscovery for [EMOApp02].Process1 Check if the return value of Deterministic-Client::WaitForNextActivation is kServiceDiscovery for [EMOApp02].Process2
Step 9	[EMOApp02] Observe the return value of DeterministicClient::WaitForNextActivation.	Check if the return value of Deterministic-Client::WaitForNextActivation is kInit for [EMOApp02].Process1 Check if the return value of Deterministic-Client::WaitForNextActivation is kInit for [EMOApp02].Process2

▽

△

Step 10	[EMOApp02] Observe the return value of DeterministicClient::WaitForNextActivation.	Check if the return value of Deterministic-Client::WaitForNextActivation is kRun for [EMOApp02].Process1 Check if the return value of Deterministic-Client::WaitForNextActivation is kRun for [EMOApp02].Process2
Step 11	[EMOApp02] Invoke DeterministicClient::RunWorkerPool API with parameter runnable object which holds the worker_runnable method and another parameter which holds container elements	
Step 12	[EMOApp02] Invoke GetRandom() API for container element <*ContElem1*> within worker_runnable method	check if the returned random number is <*RandNum1*> for [EMOApp02].Process1 check if the returned random number is <*RandNum1*> for [EMOApp02].Process2
Step 13	[EMOApp02] Invoke GetRandom() API for container element <*ContElem2*> within worker_runnable method	check if the returned random number is <*RandNum2*> for [EMOApp02].Process1 check if the returned random number is <*RandNum2*> for [EMOApp02].Process2
Step 14	[EMOApp02] Invoke DeterministicClient::GetActivationTime API	check if the time value is <TimeVal1_1> for [EMOApp02].Process1 check if the time value is <TimeVal1_1> for [EMOApp02].Process2
Step 15	[EMOApp02] Invoke DeterministicClient::GetNextActivationTime API	check if the time value is <TimeVal1_2> for [EMOApp02].Process1 check if the time value is <TimeVal1_2> for [EMOApp02].Process2

6 Test configuration and test steps for Diagnostics

6.1 Test System

6.1.1 Test configurations

6.1.1.1 STC_DIAG_00001

Configuration ID	STC_DIAG_00001
Description	Standard Jenkins server for diagnostic test
ECU 1	Hardware, 192.168.100.5
Jenkins	Jenkins Server, 192.168.100.10

The Jenkins Server running the job with the [Diagnostic Tester] is connected via Ethernet to [ECU1] hosting the System Test Application [DIAGApp01] respectively. The [Diagnostic Tester] will open TCP connections on port 13400 and send diagnostic data as UDS requests in DoIP packets.

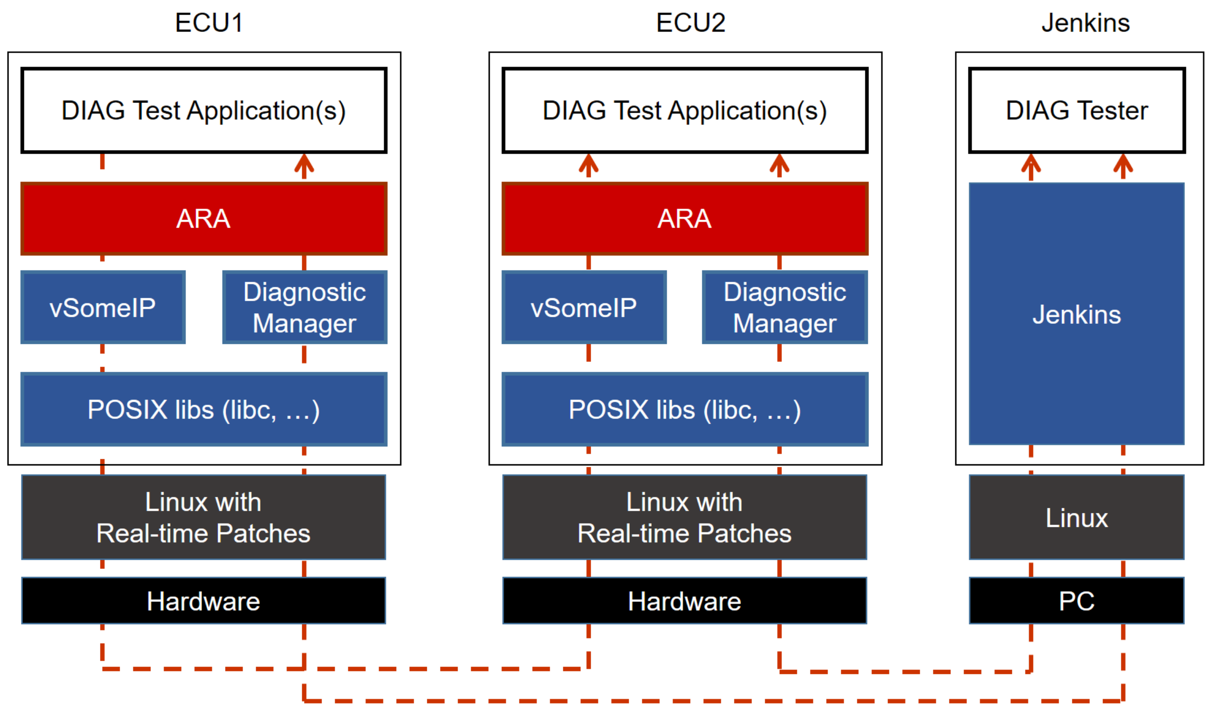

Figure 6.1: Illustration of test setup for Diagnostics.

6.1.1.2 STC_DIAG_00002

Configuration ID	STC_DIAG_00002
Description	Standard Jenkins server for diagnostic test
ECU 1	Hardware, 192.168.100.5
Jenkins	Jenkins Server, 192.168.100.10

The Jenkins Server running the job with the [Diagnostic Tester] is connected via Ethernet to [ECU1] hosting the System Test Application [DIAGApp01] respectively. The [Diagnostic Tester] will open TCP connections on port 13400 and send diagnostic data as UDS requests in DoIP packets.

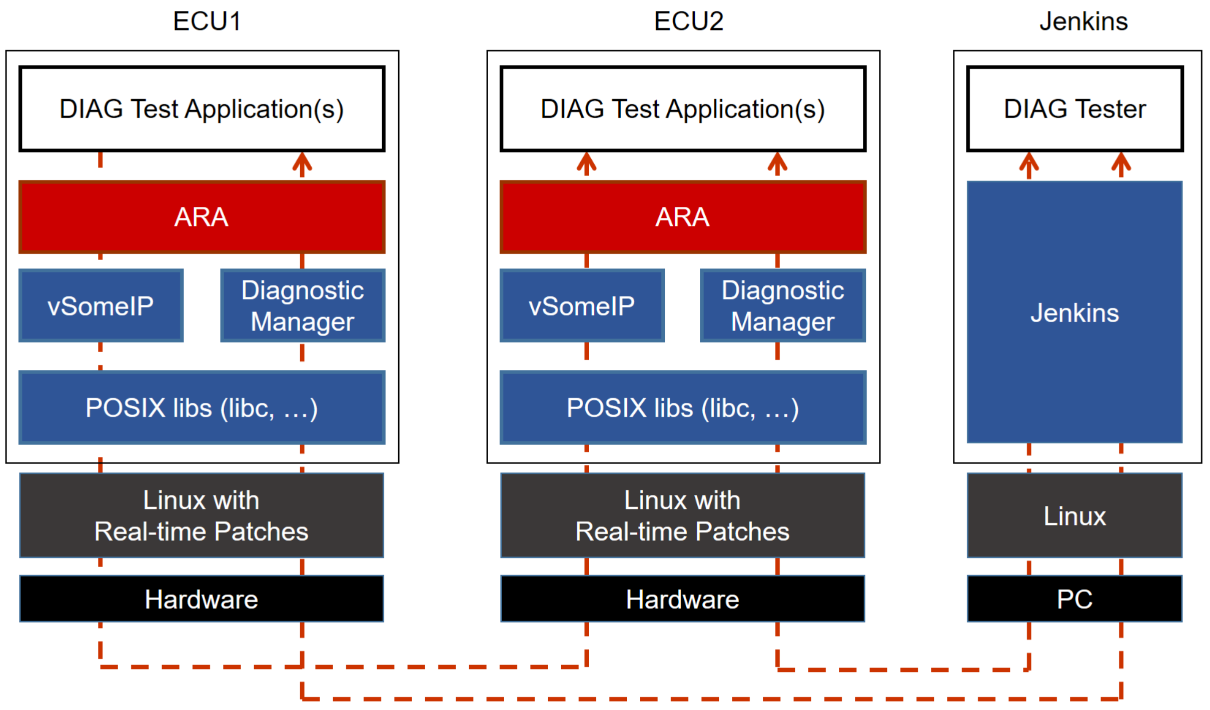

Figure 6.2: Illustration of test setup for Diagnostics.

DEM Configuration Parameters :

- DiagnosticMemoryDestination should be configured for the DTC

- DiagnosticMemoryDestination.typeOfFreezeFrameRecordNumeration should be set - to "Calculated"

- DiagnosticEnableCondition should be configured for DiagnosticEvent

- DiagnosticCommonProps.memoryEntryStorageTrigger should be configured to "con - firmed"

- DiagnosticTroubleCodeProps.freezeFrame reference should exists

- DiagnosticTroubleCodeProps.maxNumberFreezeFrameRecords should be "1"

- DiagnosticTroubleCodeProps.snapshotRecordContent should be configured

- DiagnosticFreezeFrame.trigger should be "confirmed"

- DiagnosticFreezeFrame.recordNumber should be configured to "1"

- DiagnosticFreezeFrame.update should be "true"

- OperationCycle should be configured

- DiagnosticOperationCycle.cycleAutostart should be configured as "false"

- DiagnosticOperationCycle.automaticEnd should be configured as "false"

- DiagnosticOperationCycle.cycleStatusStorage should be configured as "false

- DiagnosticEvent.eventClearAllowed should be configured as "always"

- DiagnosticEvent.clearEventBehavior should be configured as "onlyThisCycleAndRea
- diness"

- DiagnosticEvent.recoverableInSameOperationCycle should be configured as "true"

- <1000> service instance should be configured for DiagnosticOperationCycleInterface

- <1001> service instance should be configured for DiagnosticConditionInterface

- <1002> service instance should be configured for DiagnosticDTCInformationInterface

- <1003> service instance should be configured for DiagnosticMonitorInterface

- <1004> service instance should be configured for DiagnosticEventInterface

6.2 Test cases

6.2.1 [STS_DIAG_00001] Utilization of Diagnostic service ReadDataByIdentifier (0x22) by external Tester via UDS messages over DoIP.

Test Objective	Verification of correct behavior of Diagnostic service ReadDataByIdentifier (0x22) by external Tester via UDS messages over DoIP.		
ID	STS_DIAG_00001	**State**	Draft
Affected Functional Cluster	Diagnostic		
Trace to RS Criteria	RS_Diag_04196, RS_Diag_04203, RS_Diag_04172		
Reference to Test Environment	STC_DIAG_00001		

▽

△

Configuration Parameters	- Diagnostics module: • Service instance for service ReadDataByIdentifier with DID <0x0001> is configured. • Service instance with DID <0x0099> is NOT configured.	
Summary	This basic test tries to query the value of a variable contained by [DIAGApp01] on [ECU1] over the AP Diagnostics Module. The UDS service ReadDataByIdentifier (0x22) is used. The AP Diagnostics Module has to call a service in the Application Layer to retrieve the requested information and send it back as UDS response. If an unknown identifier is queried, a negative response must be sent.	
Pre-conditions	- [Diagnostic Tester] is connected to [ECU1] via TCP socket on DoIP-Port. - Software components on [ECU1] are initialized.	
Post-conditions	TCP connection between [Diagnostic Tester] and [ECU1] is closed.	
Main Test Execution		
Test Steps		**Pass Criteria**
Step 1	[Diagnostic Tester] Send Routing Activation Request (0x00005) with Activation type : Default(0x00)	
Step 2	[DIAGApp01] Send Routing Activation Response	
Step 3	[Diagnostic Tester] Send UDS Request to query value of <int1>: UDS-Service: ReadDataByIdentifier UDS-Payload: 0x22 ...	
Step 4	[DIAGApp01] Start mechanism to read the value of <int1>.	
Step 5	[Diagnostic Tester] Receive UDS response and save value of <int1> in <var1>.	Positive response received (0x62 ...). Payload of UDS response contains DID data with value of <int1>.
Step 6	[DIAGApp01] Start mechanism to change the value of <int1> by <delta>.	
Step 7	[Diagnostic Tester] Send UDS Request to query value of <int1>: UDS-Service: ReadDataByIdentifier UDS-Payload: 0x22 ...	
Step 8	[DIAGApp01] Start mechanism to read value of <int1> and return it as DID data.	
Step 9	[Diagnostic Tester] Receive UDS response and save value of <int1> in <var2>.	Positive response received (0x62 ...). Payload of UDS response contains DID data. Compare values of <var1> and <var2>. <var2> should be greater than <var1> by <delta> i.e. <var2>=<var1> + <delta>.
Step 10	[Diagnostic Tester] Send UDS Request to query data with a non-implemented DID: UDS-Service: ReadDataByIdentifier UDS-Payload: 0x22 ...	Tester receives negative response: 0x7F 0x22 0x31.

6.2.2 [STS_DIAG_00002] Utilization of Diagnostic service RoutineControl (0x31) by external Tester via UDS messages over DoIP.

Test Objective	Verification of correct behavior of Diagnostic service RoutineControl (0x31) by external Tester via UDS messages over DoIP.		
ID	STS_DIAG_00002	**State**	Draft
Affected Functional Cluster	Diagnostic		
Trace to RS Criteria	RS_Diag_04224, RS_Diag_04196, RS_Diag_04203, RS_Diag_04006 RS_Diag_04172		
Reference to Test Environment	STC_DIAG_00001		
Configuration Parameters	- The following service is configured [DIAGService01] in [DIAGApp01] - In this [DIAGService01], two different contents are available • <Content1> • <Content2> - Diagnostics module: • Service instance for service RoutineControl with RID <0x0001> is configured and only available in Extended Diagnostic Session. • Service Diagnostic Session Control is configured.		
Summary	This test tries to start a routine in [DIAGApp01] over the AP Diagnostics Module and the UDS service RoutineControl (0x31). In DefaultSession, execution is not allowed and a negative response is sent. After switching to ExtendedDiagnosticSession, the routine is started and a positive response is sent.		
Pre-conditions	- [Diagnostic Tester] is connected to [ECU1] via TCP socket on DoIP-Port. - Software components on [ECU1] are initialized. - [DIAGApp01] sends <Content1> via [DIAGService01].		
Post-conditions	TCP connection between Jenkins server and [ECU1] is closed.		
Main Test Execution			
Test Steps			**Pass Criteria**
Step 1	[Diagnostic Tester] Send Routing Activation Request (0x00005) with Activation type : Default(0x00)		
Step 2	[DIAGApp01] Send Routing Activation Response		
Step 3	[Diagnostic Tester] Send UDS request to change content of [DIAGService01]: UDS-Service: RoutineControl UDS-Payload: 0x31 0x01 ...		Negative response received: Service Not Supported in Active Session (0x7F 0x31 0x7F).
Step 4	[Diagnostic Tester] Send UDS request to start an Extended Diagnostic Session: UDS-Service: DiagnosticSessionControl UDS-Payload: 0x10 0x03		Positive response received (0x50 0x03).
Step 5	[Diagnostic Tester] Send UDS request to change content of [DIAGService01] from <Content1> to <Content2>: UDS-Service: RoutineControl UDS-Payload: 0x31 0x01 ...		

▽

△

Step 6	[DIAGApp01] Start mechanism to change content of [DIAGService01] from <Content1> to <Content2>	Content of Service is changed to <Content2>
Step 7	[DIAGApp01] Return from Subfunction Start of Routine with RID <0x0001>.	
Step 8	[Diagnostic Tester] Receive UDS response.	Positive response received (0x71 ...).
Step 9	[Diagnostic Tester] Send UDS request to start an Default Diagnostic Session: UDS-Service: DiagnosticSessionControl UDS-Payload: 0x10 0x01	Positive response received (0x50 0x01).
Step 10	[Diagnostic Tester] Send UDS request to start an Invalid Diagnostic Session: UDS-Service: DiagnosticSessionControl UDS-Payload: 0x10 0x50	Negative response sub-functionNotSupported is received (0x7F 0x10 0x12).
Step 11	[Diagnostic Tester] Send UDS request to change content of [DIAGService01]: UDS-Service: RoutineControl UDS-Payload: 0x31 0x01 ...	Negative response received: Service Not Supported in Active Session (0x7F 0x31 0x7F).

6.2.3 [STS_DIAG_00003] Utilization of Diagnostic service TesterPresent (0x3E) by External Tester via UDS messages over DoIP.

Test Objective	Verification of correct behavior of Diagnostic service TesterPresent (0x3E) by External Tester via UDS messages over DoIP.		
ID	STS_DIAG_00003	**State**	Draft
Affected Functional Cluster	Diagnostic		
Trace to RS Criteria	RS_Diag_04196, RS_Diag_04203, RS_Diag_04006, RS_Diag_04020		
Reference to Test Environment	STC_DIAG_00001		
Configuration Parameters	Diagnostics module: • Service instance for service RoutineControl with RID <0x0001> is configured and only available in Extended Diagnostic Session. • Service Diagnostic Session Control and Extended Diagnostic Session time out is configured. • TesterPresent is configured.		
Summary	TesterPresent request is sent to indicate that previously activated non-default (e.g. extended) session will still be active. The UDS service RoutineControl (0x31) is executed to check if Extended session is active (Any other service which is supported in extended session may be used). Positive response is received for the TesterPresent request if suppressPosRspMsgIndicationBit is set to FALSE. No response is expected (by Client) from Server if, suppressPosRspMsgIndicationBit is set to TRUE		

▽

△

Pre-conditions	- [Diagnostic Tester] is connected to [ECU1] via TCP socket on DoIP-Port. - Software components on [ECU1] are initialized.	
Post-conditions	TCP connection between Jenkins server and [ECU1] is closed.	
Main Test Execution		
Test Steps		**Pass Criteria**
Step 1	[Diagnostic Tester] Send Routing Activation Request (0x00005) with Activation type : Default(0x00)	
Step 2	[DIAGApp01] Send Routing Activation Response	
Step 3	[Diagnostic Tester] Send UDS request to start an Extended Diagnostic Session: UDS-Service: DiagnosticSessionControl(SID 0x10) UDS-Payload: 0x10 0x03	Positive response received (0x50 0x03).
Step 4	[Diagnostic Tester] Wait for time <t1> such that <t1> is less than Diagnostic session timer timeout.	
Step 5	[Diagnostic Tester] Send UDS request Tester Present with suppressPosRspMsg IndicationBit is set to FALSE. UDS-Service: TesterPresent (SID 0x3E) UDS-Payload: 0x3E 0x00	Positive response received (0x7E 0x00).
Step 6	[Diagnostic Tester] Wait for time <t2> such that - 1) <t2> is greater than Diagnostic session timer timeout. 2) <t2> is less than sum of Extended session timer and Diagnostic session timer timeout.	
Step 7	[Diagnostic Tester] Send UDS request RoutineControl to confirm if Extended Session is active. UDS-Service: RoutineControl (SID 0x31) UDS-Payload: 0x31 0x01 ...	Positive response received (0x71 ...).
Step 8	[Diagnostic Tester] Stop sending TesterPresent and wait for Extended Diagnostic Session to time out	
Step 9	[Diagnostic Tester] Send UDS request RoutineControl to confirm if Extended Session is active. UDS-Service: RoutineControl UDS-Payload: 0x31 0x01 ...	Negative response received: Service Not Supported in Active Session (0x7F 0x31 0x7F (NRC)).
Step 10	[Diagnostic Tester] Send UDS request to start an Extended Diagnostic Session: UDS-Service: DiagnosticSessionControl UDS-Payload: 0x10 0x03	Positive response received (0x50 0x03).
Step 11	[Diagnostic Tester] Wait for time <t1> such that <t1> is less than Diagnostic session timer timeout.	

▽

△

Step 12	[Diagnostic Tester] Send UDS request TesterPresent with suppressPosRspMsg IndicationBit is set to TRUE. UDS-Service: TesterPresent UDS-Payload: 0x3E 0x80	No response received for UDS request TesterPresent.
Step 13	[Diagnostic Tester] Wait for time <t2> such that - 1) <t2> is greater than Diagnostic session timer timeout. 2) <t2> is less than sum of Extended session timer and Diagnostic session timer timeout.	
Step 14	[Diagnostic Tester] Send UDS request RoutineControl to confirm if Extended Session is active. UDS-Service: RoutineControl UDS-Payload: 0x31 ...	Positive response received (0x71 ...).

6.2.4 [STS_DIAG_00004] Utilization of Diagnostic service WriteDataByIdentifier (0x2E) by External Tester via UDS messages over DoIP.

Test Objective	Verification of correct behavior of Diagnostic service WriteDataByIdentifier (0x2E) by External Tester via UDS messages over DoIP.		
ID	STS_DIAG_00004	**State**	Draft
Affected Functional Cluster	Diagnostic		
Trace to RS Criteria	RS_Diag_04196, RS_Diag_04203, RS_Diag_04172		
Reference to Test Environment	STC_DIAG_00001		
Configuration Parameters	Diagnostics module: - Service instances for service ReadDataByIdentifier and WriteDataByIdentifier with DID <0x0001> are configured.		
Summary	This basic test tries to query the value of <int1> contained by [DIAGApp01] on [ECU1] over the AP Diagnostics Module. The UDS service ReadDataByIdentifier (0x22) is used and then the value of <int1> is overwritten by UDS service WriteDataByIdentifier (0x2E). Overwritten value of the variable <int1> is read back using UDS service ReadDataByIdentifier (0x22).		
Pre-conditions	- [Diagnostic Tester] is connected to [ECU1] via TCP socket on DoIP-Port - Software components on [ECU1] are initialized.		
Post-conditions	TCP connection between [Diagnostic Tester] and [ECU1] is closed.		
Main Test Execution			
Test Steps			**Pass Criteria**
Step 1	[Diagnostic Tester] Send Routing Activation Request (0x00005) with Activation type : Default(0x00)		
Step 2	[DIAGApp01] Send Routing Activation Response		

▽

△

Step 3	[Diagnostic Tester] Send UDS Request to query value of <int1>: UDS-Service: ReadDataByIdentifier UDS-Payload: 0x22 ...	
Step 4	[DIAGApp01] Wait for invocation.	Implementation of method Read for DID <0x0001> is invoked.
Step 5	[Diagnostic Tester] Receive UDS response with value of <int1>.	Positive response received (0x62 ...). Payload of UDS response contains DID data with value of <int1>.
Step 6	[Diagnostic Tester] Send UDS Request to overwrite value of <int1> with <int2> UDS-Service: WriteDataByIdentifier UDS-Payload: 0x2E ...	
Step 7	[Diagnostic Tester] Receive UDS response.	Positive response received (0x6E ...) after successful write.
Step 8	[Diagnostic Tester] Send UDS request to query value of <int1> UDS-Service: ReadDataByIdentifier UDS-Payload: 0x22 ...	
Step 9	[DIAGApp01] Wait for invocation.	Implementation of method Read for DID <0x0001> is invoked.
Step 10	[Diagnostic Tester] Receive UDS response with value of <int1> and store it in <var>.	Positive response received (0x62 ...). Payload of UDS response contains DID data with value of <int1>.
Step 11	[Diagnostic Tester] Compare <var> and <int2> values.	Both values should be equal.

6.2.5 [STS_DIAG_00005] Utilization of Diagnostic service InputOutputControlByIdentifier (0x2F) by External Tester via UDS messages over DoIP.

Test Objective	Verification of correct behavior of Diagnostic service InputOutputControlByIdentifier (0x2F) by External Tester via UDS messages over DoIP.		
ID	STS_DIAG_00004	**State**	Draft
Affected Functional Cluster	Diagnostic		
Trace to RS Criteria	RS_Diag_04218, RS_Diag_04172		
Reference to Test Environment	STC_DIAG_00001		

▽

△

Configuration Parameters	Diagnostics module: - Service instances for service InputOutputControlByIdentifier with DID <0x0001> are configured. - Methods ShortTermAdjustment , FreezeCurrentState ,ReturnControlToECU ,ResettoDefault for InputOutputControlByIdentifier for DID <0x001>are available	
Summary	This basic test tries to send request for ShortTermAdjustment/FreezeCurrentState/ResettoDefault/FreezeCurrentState for DID <0x001> contained by [DIAGApp01]on [ECU1] over the AP Diagnostics Module. This test tries to substitute values of the input for DID <0x0001> and verify the output as desired	
Pre-conditions	- [Diagnostic Tester] is connected to [ECU1] via TCP socket on DoIP-Port - Software components on [ECU1] are initialized.	
Post-conditions	TCP connection between [Diagnostic Tester] and [ECU1] is closed.	
Main Test Execution		
Test Steps		**Pass Criteria**
Step 1	[Diagnostic Tester] Send Routing Activation Request (0x00005) with Activation type : Default(0x00)	
Step 2	[DIAGApp01] Send Routing Activation Response	
Step 3	[Diagnostic Tester] Send UDS Request for ShortTermAdjustment to value <x> for DID <0x0001> SID :0x2F ,InputOutputcontrolParameter = 0x03(ShortTermAdjustment) Payload : 0x2F 0x00 0x01 03 ...	
Step 4	[DIAGApp01] Wait for invocation.	Implementation of method ShortTermAdjustment for DID <0x0001> is invoked.
Step 5	[Diagnostic Tester] Receive UDS response with desired ShortTermAdjustment	Positive response received (0x6F ...). Payload of UDS response contains DID data with desired shorttermadjustment.
Step 6	[Diagnostic Tester] Send UDS Request to Freeze State of DID<0x001> SID :0x2F ,InputOutputcontrolParameter = 0x02(FreezeCurrentState) UDS-Payload: 0x2F ...	
Step 7	[DIAGApp01] Wait for invocation.	Implementation of method FreezeCurrentState for DID <0x0001> is invoked.
Step 8	[Diagnostic Tester] Receive UDS response with Current State Freezed.	Positive response received (0x6F ...). Payload of UDS response contains DID data .
Step 9	[Diagnostic Tester] Send UDS request to ResetToDefault SID :0x2F ,InputOutputcontrolParameter = 0x01(ResetToDefault) UDS-Payload: 0x2F ...	
Step 10	[DIAGApp01] Wait for invocation.	Implementation of method ResetToDefault for DID <0x0001> is invoked.
Step 11	[Diagnostic Tester] Receive UDS response	Positive response received (0x6F ...). Payload of UDS response contains DID data reset to default .

6.2.6 [STS_DIAG_00006] Utilization of Diagnostic service ClearDTC (0x14) by External Tester via UDS messages over DoIP.

Test Objective	Verification of correct behavior of Diagnostic service ClearDTC (0x14) by External Tester via UDS messages over DoIP.		
ID	STS_DIAG_00006	**State**	Draft
Affected Functional Cluster	Diagnostic		
Trace to RS Criteria	RS_Diag_04196, RS_Diag_04203		
Reference to Test Environment	STC_DIAG_00001		
Configuration Parameters	Diagnostics module: - Service instances for service Clear DTC(0x14) are configured. - GroupofDTC <gtc1> is configured.		
Summary			
Pre-conditions	- [Diagnostic Tester] is connected to [ECU1] via TCP socket on DoIP-Port - Software components on [ECU1] are initialized.		
Post-conditions	TCP connection between [Diagnostic Tester] and [ECU1] is closed.		
Main Test Execution			
Test Steps		**Pass Criteria**	
Step 1	[Diagnostic Tester] Send Routing Activation Request (0x00005) with Activation type : Default(0x00)		
Step 2	[DIAGApp01] Send Routing Activation Response		
Step 3	[Diagnostic Tester] Send UDS request to clear GroupofDTC<gtc1> related to event <e1> SID :0x14 Payload : 0x14 0xFF 0xFF 0x33		
Step 4	[DIAGApp01] Implementation of Service Clear DTC is invoked.	Check if requested GroupofDTC<gtc1> is present in the configured group of DTC. If yes, Send response.	
Step 5	[Diagnostic Tester] Receive UDS response	Positive response received (0x54 ...). Payload of UDS response contains status of cleared DTC.	
Step 6	[Diagnostic Tester] Send UDS request to read cleared GroupofDTC<gtc1> related to event <e1> SID :0x19 Payload : 0x19 ..		
Step 7	[DIAGApp01] Invoke implementation of Diagnostic Service Read DTC	Check if DTC is available.	
Step 8	[Diagnostic Tester] Receive UDS response	Positive response (0x59)with no available DTC is received	

▽

△

Step 9	[Diagnostic Tester] Send UDS request to clear GroupofDTC<gtc1> related to event <e1> SID :0x14 Payload: 0x14 0xFF FF .	
Step 10	[DIAGApp01] Implementation of service Clear DTC is invoked.Check Length of requested request	If length of requested UDS request is incorrect send NRC-13.
Step 11	[Diagnostic Tester] Receive UDS response for Clear DTC.	Negative response received (0x7F 0x14 0x13...).
Step 12	[Diagnostic Tester] Send UDS request for session change SID : 0x10 Payload: 0x10 0x03	
Step 13	[DIAGApp01] Prepare to start session change to extended session	
Step 14	[DiagnosticTester] Receive positive response for session change SID :0x10 Payload : 0x50 0x03	
Step 15	[Diagnostic Tester] Send UDS request to clear GroupofDTC<gtc1> related to event <e1> SID : 0x14 Payload: 0x14 0xFF 0xFF 0x35	
Step 16	[DIAGApp01] Implementation of service Clear DTC is invoked.Check if requested DTC group is available.	Group of DTC is not available,Send NRC-31 .
Step 17	[Diagnostic Tester] Receive UDS response	Negative response received (0x7F 0x14 0x31...)

6.2.7 [STS_DIAG_00007] Utilization of Diagnostic service SecurityAccess (0x27) by External Tester via UDS messages over DoIP.

Test Objective	Verification of correct behavior of Diagnostic service SecurityAccess (0x27) by External Tester via UDS messages over DoIP.		
ID	STS_DIAG_00007	**State**	Draft
Affected Functional Cluster	Diagnostic		
Trace to RS Criteria	RS_Diag_04005, RS_Diag_04172		

▽

△

Reference to Test Environment	STC_DIAG_00001	
Configuration Parameters	Diagnostics module: - Service instances for service Security access are configured - Service instances for Service ReadDataByIdentifier with DID <0x0001> are configured. - Sub functions (SecurityAccessType) are configured.	
Summary	This basic test tries to get an access of an ECU using Diagnostic service Security Access and try to access some secured parameters (DID <0x0001>)of an ECU. Tester first request for SEED, ECU responds with the SEED Value(random 2 byte number). Tester then generates the Key using the received SEED(Lower nibble of each byte masked with 0 ,Note that this could be OEM specific we are considering this as an example for demonstration) and send it to an ECU.ECU then verifies the key and grants access (Positive Response) .If Length of the request /sub function is not supported, then ECU shall send NRC	
Pre-conditions	- [Diagnostic Tester] is connected to [ECU1] via TCP socket on DoIP-Port - Software components on [ECU1] are initialized.	
Post-conditions	TCP connection between [Diagnostic Tester] and [ECU1] is closed.	
Main Test Execution		
Test Steps		**Pass Criteria**
Step 1	[Diagnostic Tester] Send Routing Activation Request (0x00005) with Activation type : Default(0x00)	
Step 2	[DIAGApp01] Send Routing Activation Response	
Step 3	[Diagnostic Tester] Send UDS request to gain SecurityAccessType - 1 SID : 0x27 Payload - 0x27 01 ..	
Step 4	[DIAGApp01] Implementation of method RequestSeed is invoked	Seed (2 bytes of random number)is generated successfully and response is sent
Step 5	[Diagnostic Tester] Send Request to SendKey SID: 0x27 Payload : 0x27 0x02 ...	
Step 6	[DIAGApp01] Invoke method to verify received key	Check if the received Key is equal to internally generated key ,if yes send positive response
Step 7	[Diagnostic Tester] Receive positive response.	Positive response (0x67 ..) is received
Step 8	[Diagnostic Tester] Send Request to read a secured paramter <var1> using ReadDID Service SID : 0x22 Payload : 0x22 0x00 0x01	
Step 9	[DIAGApp01] Invoke Service ReadDataByIdentifier	Provide value of <var1> as a response
Step 10	[DiagnosticTester] Receive UDS Service response	Positive response (0x62 0x00 0x01 var1)

▽

△

Step 11	[Diagnostic Tester] Send UDS request to gain SecurityAccessType -1 SID : 0x27 Payload - 0x27 01 ..	
Step 12	[DIAGApp01] Implementation of Method - RequestSeed is invoked.	Check the length of the UDS security request, if the length is not correct send NRC-13
Step 13	[Diagnostic Tester] Receive UDS response	Negative response received (0x7F 0x27 0x13 ...)
Step 14	[Diagnostic Tester] Send UDS request to gain SecurityAccessType - 2 SID : 0x27 Payload - 0x27 02 ..	
Step 15	[DIAGApp01] Implementation of Method - RequestSeed is invoked.	Check if the sub function (SecurityAccessType -2) is supported or not. If not send NRC-12
Step 16	[Diagnostic Tester] Receive UDS response	Negative response (0x7F 0x27 0x12)

6.2.8 [STS_DIAG_00008] Utilization of Diagnostic service ReadDTCInformation (0x19) by External Tester via UDS messages over DoIP.

Test Objective	Verification of correct behavior of Diagnostic service ReadDTCInformation (0x19) by external Tester via UDS messages over DoIP.		
ID	STS_DIAG_00008	**State**	Draft
Affected Functional Cluster	Diagnostic		
Trace to RS Criteria	RS_Diag_04190 RS_Diag_04195 RS_Diag_04201		
Reference to Test Environment	STC_DIAG_00001		
Configuration Parameters	Diagnostics module: - Service instances for service ReadDTCInformation (0x19) are configured. - Events <e1>, <e2> to <en> and corresponding DTCs are configured.		
Summary	Tester queries the DTCs and its related information by DTC Status Mask.		
Pre-conditions	- [Diagnostic Tester] is connected to [ECU1] via TCP socket on DoIP-Port - Software components on [ECU1] are initialized.		
Post-conditions	TCP connection between [Diagnostic Tester] and [ECU1] is closed.		
Main Test Execution			
Test Steps			**Pass Criteria**
Step 1	[Diagnostic Tester] Send Routing Activation Request (0x00005) with Activation type : Default(0x00)		

▽

△

Step 2	[DIAGApp01] Send Routing Activation Response	
Step 3	[Diagnostic Tester] Send UDS request to report number of DTCs by Status Mask related to event <e1> Request is sent with Payload: SID :0x19 <reportNumberOfDTCByStatusMask> <DTCStatusMask>	
Step 4	[DIAGApp01] Implementation of Service ReadDTCInformation is invoked.	Send number of DTCs in response if requested DTCs with status mask is present.
Step 5	[Diagnostic Tester] Receive UDS response	Positive response is received with Payload: 0x59 <reportNumberOfDTCByStatus-Mask> <DTCStatusAvailabilityMask> <DTCFormatIdentifier> <DTCCountHighByte> <DTCCountLowByte>
Step 6	[Diagnostic Tester] Send UDS request to report DTCs by Status Mask related to event <e1> Request is sent with Payload: SID :0x19 <reportDTCByStatusMask> <DTCStatusMask>	
Step 7	[DIAGApp01] Implementation of Service ReadDTCInformation is invoked.	Send list of DTCs as response if requested DTCs with status masks are present.
Step 8	[Diagnostic Tester] Receive UDS response	Positive response is received with Payload: 0x59 <reportDTCByStatusMask> <DTCStatusAvailabilityMask> <DTCHighByte> <DTCMiddleByte> <DTCLowByte> <statusOfDTC>
Step 9	[Diagnostic Tester] Send UDS request to report DTC Snapshot Identification related to event <e1> Request is sent with Payload: SID : 0x19 <reportDTCSnapshotIdentification> <DTCStatusMask>	

▽

△

Step 10	[DIAGApp01] Implementation of Service ReadDTCInformation is invoked.	Send list of DTCs along with Snapshot Record Number as response if requested DTCs with DTC Snapshot Record Number are present
Step 11	[Diagnostic Tester] Receive UDS response	Positive response is received with Payload: 0x59 <reportDTCSnapshotIdentification> <DTCStatusAvailabilityMask> <DTCHighByte> <DTCMiddleByte> <DTCLowByte> <DTCSnapshotRecordNumber>
Step 12	[Diagnostic Tester] Send UDS request to report DTC Snapshot Record by DTC Number related to event <e1> Request is sent with Payload: SID : 0x19 <reportDTCSnapshotRecordByDTCNumber> <DTCStatusMask> <DTCHighByte> <DTCMiddleByte> <DTCLowByte> <DTCSnapshotRecordNumber>	
Step 13	[DIAGApp01] Implementation of Service ReadDTCInformation is invoked.	Send DTCs with DTC Snapshot Record information as response if requested DTCs with DTC Snapshot Record information are present.
Step 14	[Diagnostic Tester] Receive UDS response	Positive response is received with Payload: 0x59 <reportDTCSnapshotRecordByDTCNumber> <DTCStatusAvailabilityMask> <DTCHighByte> <DTCMiddleByte> <DTCLowByte> <statusOfDTC> <DTCSnapshotRecordNumber> <DTCSnapshotRecordNumberOfIdentifiers> <dataIdentifierMSB> <dataIdentifierLSB> < DTCSnapshotRecordData 1> < DTCSnapshotRecordData n>

6.2.9 [STS_DIAG_00009] Storing and Reading of DTC status and snapshot data.

Test Objective	Storing and Reading of DTC status and snapshot data in the primary fault memory defined by ISO 14229-1.		
ID	STS_DIAG_00009	**State**	Draft
Affected Functional Cluster	Diagnostic		
Trace to RS Criteria	RS_Diag_04178, RS_Diag_04186, RS_Diag_04148, RS_Diag_04183, RS_Diag_04151, RS_Diag_04150, RS_Diag_04127, RS_Diag_04136,		
Reference to Test Environment	STC_DIAG_00002		
Configuration Parameters	Diagnostics module: 1. DiagnosticMonitor should be configured for DiagnosticEvent <Event0> 2. DTC should be configured for the DiagnosticEvent <Event0> 3. agingAllowed should be "false" 4. DiagnosticTroubleCodeUds.udsDtcValue should be configured as "1" 5. DiagnosticEvent.eventFailureCycleCounterThreshold should be configured as "127"		
Summary	This test case covers Reporting of Event from DiagnosticMonitor Application, Notification of EventStatus change, Notification of DTCStatus change, Setting of OperationCycle, Setting of enable condition, Notification about changing state of enable condition, Getting DTC and Event status, Notification about snapshot data change, Reading DTC status and Snapshot data by using ReadDTCInformation service 0x19.		
Pre-conditions	- [Diagnostic Tester] is connected to [ECU1] via TCP socket on DoIP-Port - Software components on [ECU1] are initialized. - Proxies should be available for DiagnosticOperationCycleInterface, DiagnosticConditionInterface, DiagnosticDTCInformationInterface, DiagnosticMonitorInterface and DiagnosticEventInterface		
Post-conditions	TCP connection between [Diagnostic Tester] and [ECU1] is closed.		
Main Test Execution			
Test Steps			**Pass Criteria**
Step 1	[Diagnostic Tester] Send Routing Activation Request (0x00005) with Activation type : Default(0x00).		
Step 2	[DIAGApp01] Send Routing Activation Response.		
Step 3	[DIAGApp01] Call SetOperationCycle with "kOperationCycleStart" for "Event0".		[DIAGApp01] SetNotifier() of DiagnosticOperationCycleInterface method should be called with kOperationCycleStart. [DIAGApp01] SetDTCStatusChangedNotifier() should be called. [DIAGApp01] SetEventStatusChangedNotifier() should be called.
Step 4	[DIAGApp01] Call GetEventStatus.		[DIAGApp01] It should return EventStatusByte as 0x40.

▽

△

Step 5	[DIAGApp01] Call GetCurrentStatus.	[DIAGApp01] It should return UdsDtcStatusBitType as 0x50.
Step 6	[DIAGApp01] Call SetCondition with "kConditionTrue" for "Event0"	[DIAGApp01] InitMonitorReason() should be called with kReenabled.
Step 7	[DIAGApp01] Call GetCondition for "Event0".	[DIAGApp01] It should return 0x01.
Step 8	[DIAGApp01] Call ReportMonitorAction with MonitorAction as kFailed from DiagnosticMonitor Application.	[DIAGApp01] InitMonitorReason() should be called with MonitorAction as kFailed. [DIAGApp01] SetDTCStatusChangedNotifier() should be called. [DIAGApp01] SetEventStatusChangedNotifier() should be called. [DIAGApp01] SetSnapshotRecordUpdatedNotifier() should be called for snapShotData Change for DID 1.
Step 9	[DIAGApp01] Call GetEventStatus.	[DIAGApp01] It should return EventStatusByte as 0x03.
Step 10	[DIAGApp01] Call GetCurrentStatus	[DIAGApp01] It should return UdsDtcStatusBitType as 0x2F
Step 11	[Diagnostic Tester] Call ReadDTCInformation (0x19) for reading snapShotData of DID 1 19 04 0xFF.	[DiagnosticManager] It should return stored DTC status and SnapShot data of DID 1.

6.2.10 [STS_DIAG_00010] Control of DTC storage via UDS service 0x85.

Test Objective	The diagnostic in AUTOSAR shall support control of DTC storage via UDS service 0x85.		
ID	STS_DIAG_00010	**State**	Draft
Affected Functional Cluster	Diagnostic		
Trace to RS Criteria	RS_Diag_04159		
Reference to Test Environment	STC_DIAG_00002		

▽

△

Configuration Parameters	Diagnostics module: 1. DiagnosticMonitor should be configured for DiagnosticEvent <Event0> 2. DTC should be configured for the DiagnosticEvent <Event0> 3. agingAllowed should be "false" 4. DiagnosticTroubleCodeUds.udsDtcValue should be configured as "1" 5. DiagnosticEvent.eventFailureCycleCounterThreshold should be configured as "127"	
Summary	This test case covers functionality of service 0x85 and Re-enabling of ControlDTCSettings by calling EnableControlDtc.	
Pre-conditions	- [Diagnostic Tester] is connected to [ECU1] via TCP socket on DoIP-Port - Software components on [ECU1] are initialized.	
Post-conditions	TCP connection between [Diagnostic Tester] and [ECU1] is closed.	
Main Test Execution		
Test Steps		**Pass Criteria**
Step 1	[Diagnostic Tester] Send Routing Activation Request (0x00005) with Activation type : Default(0x00).	
Step 2	[DIAGApp01] Send Routing Activation Response.	
Step 3	[Diagnostic Tester] Request for service 0x85 (ControlDTCSetting) 0x85 0x01 0xFFFFFF.	[DIAGApp01] SetControlDtcStatusNotifier should be called after changing the ControlDTCSetting. [Diagnostic Tester] DM should send positive response as 0xC5 0x001.
Step 4	[DIAGApp01] Call GetControlDTCStatus.	[DIAGApp01] GetControlDTCStatus should return DTC status as kDTCSettingOn.
Step 5	[Diagnostice Tester] Request for service 0x85 (ControlDTCSetting) 0x85 0x02 0xFFFFFF.	[DIAGApp01] SetControlDtcStatusNotifier should be called after changing the ControlDTCSetting.
Step 6	[DIAGApp01] Call GetControlDTCStatus.	[DIAGApp01] GetControlDTCStatus should return DTC status as kDTCSettingOff.
Step 7	[DIAGApp01] Call EnableControlDtc.	[DIAGApp01] SetControlDtcStatusNotifier should be called after changing the ControlDTCSetting.
Step 8	[DIAGApp01] Call GetControlDTCStatus.	[DIAGApp01] GetControlDTCStatus should return DTC status as kDTCSettingOn.

6.2.11 [STS_DIAG_00011] Provide connection specific meta information to external service processors.

Test Objective	The diagnostic in AUTOSAR shall provide connection specific meta-information to the external service processor, which is processing the UDS service request. At least DoIP shall be supported and the meta-information shall contain Src-IP-Adr/Port and Target-IP-Adr/Port of the request. The meta-information should be designed, that it can later easily extended to also cover connection information of other network technologies (like CAN, Flexray).		
ID	STS_DIAG_00011	**State**	Draft
Affected Functional Cluster	Diagnostic		
Trace to RS Criteria	RS_Diag_04170		
Reference to Test Environment	STC_DIAG_00001		
Configuration Parameters	Diagnostics module: 1. Service instance for service ReadDataByIdentifier with DID <0x0001> is configured. 2. Service instance with DID <0x0099> is NOT configured.		
Summary	Provides connection specific meta-information to external service processors		
Pre-conditions	- [Diagnostic Tester] is connected to [ECU1] via TCP socket on DoIP-Port - Software components on [ECU1] are initialized.		
Post-conditions	TCP connection between [Diagnostic Tester] and [ECU1] is closed.		
Main Test Execution			
Test Steps		**Pass Criteria**	
Step 1	[Diagnostic Tester] Send Routing Activation Request (0x00005) with Activation type : Default(0x00).		
Step 2	[DIAGApp01] Send Routing Activation Response.		
Step 3	[Diagnostic Tester] Send UDS Request to query value of <int1> UDS-Service: ReadDataByIdentifier UDS-Payload: 0x22 ...	[DIAGApp01] Application should receive the meta information containing SA, TA, Source Port, Target Port, Target Address Type, RequestHandle. [Diagnostic Tester] Positive response received (0x62 ...). Payload of UDS response contains DID data with value of <int1>.	

6.2.12 [STS_DIAG_00012] Event debounce counter shall be configurable.

Test Objective	Debounce counter should be frozen, when at least one enable condition for the event is set to "not fulfilled".		
ID	STS_DIAG_00012	**State**	Draft

▽

△

Affected Functional Cluster	Diagnostic	
Trace to RS Criteria	RS_Diag_04125	
Reference to Test Environment	STC_DIAG_00002	
Configuration Parameters	Diagnostics module: 1. DiagnosticMonitor should be configured for DiagnosticEvent "Event0" 2. DTC should be configured for the DiagnosticEvent "Event0" 3. agingAllowed should be "true" 4. DiagnosticTroubleCodeUds.udsDtcValue should be configured as "1" 5. DiagnosticEvent.eventFailureCycleCounterThreshold should be configured as "127" 6. DiagnosticAging.threshold shall be "2" 7. DiagnosticAging.agingCycle shall refer to operation cycle as "POWER" 8. DiagEventDebounceCounterBased.counterIncrementStepSize should be "64" 9. DiagEventDebounceCounterBased.counterFailedThreshold should be "2" 10. DiagnosticDebounceAlgorithmProps.debounceBehavior should be "freeze"	
Summary	This test case covers, the debounce counter shall be frozen, when at least one enable condition for the event is set to "not fulfilled" and in case of switching the enable conditions to "fulfilled" the monitor needs to be informed to restart the event detection.	
Pre-conditions	- [Diagnostic Tester] is connected to [ECU1] via TCP socket on DoIP-Port - Software components on [ECU1] are initialized. - Proxies should be available for DiagnosticOperationCycleInterface, DiagnosticConditionInterface, DiagnosticDTCInformationInterface, DiagnosticMonitorInterface and DiagnosticEventInterfac	
Post-conditions	TCP connection between [Diagnostic Tester] and [ECU1] is closed.	
Main Test Execution		
Test Steps		**Pass Criteria**
Step 1	[Diagnostic Tester] Send Routing Activation Request (0x00005) with Activation type : Default(0x00).	
Step 2	[DIAGApp01] Send Routing Activation Response.	
Step 3	[DIAGApp01] Call SetOperationCycle with "kOperationCycleStart" for "Event0"	[DIAGApp01] SetNotifier() of DiagnosticOperationCycleInterface method should be called with kOperationCycleStart.
Step 4	[DIAGApp01] Call SetCondition with "kConditionTrue" for "Event0"	[DIAGApp01] InitMonitorReason() should be called with kReenabled.
Step 5	[DIAGApp01] Call ReportMonitorAction with MonitorAction as kPrefailed from DiagnosticMonitor Application	
Step 6	[DIAGApp01] Call GetFaultDetectionCounter	[DIAGApp01] GetFaultDetectionCounter should return 64.
Step 7	[DIAGApp01] Call SetCondition with "kConditionFalse" for "Event0"	[DIAGApp01] Enable condtion state should be changed to false.

▽

△

Step 8	[DIAGApp01] Call ReportMonitorAction with MonitorAction as kPrefailed from DiagnosticMonitor Application	
Step 9	[DIAGApp01] Call GetFaultDetectionCounter	[DIAGApp01] GetFaultDetectionCounter should return 64.
Step 10	[DIAGApp01] Call SetCondition with "kConditionTrue" for "Event0"	[DIAGApp01] InitMonitorReason() should be called with kReenabled.
Step 11	[DIAGApp01] Call ReportMonitorAction with MonitorAction as kPrefailed from DiagnosticMonitor Application	
Step 12	[DIAGApp01] Call GetFaultDetectionCounter	[DIAGApp01] GetFaultDetectionCounter should return 127.

6.2.13 [STS_DIAG_00013] The diagnostic in AUTOSAR shall provide the reporting of DTCs and related data.

Test Objective	The diagnostic in AUTOSAR shall provide the reporting of DTCs and related data.		
ID	STS_DIAG_00013	**State**	Draft
Affected Functional Cluster	Diagnostic		
Trace to RS Criteria	RS_Diag_04157		
Reference to Test Environment	STC_DIAG_00002		
Configuration Parameters	Diagnostics module: 1. DiagnosticMonitor should be configured for DiagnosticEvent <Event0> 2. DTC should be configured for the DiagnosticEvent <Event0> 3. agingAllowed should be "false" 4. DiagnosticTroubleCodeUds.udsDtcValue should be configured as "1" 5. DiagnosticEvent.eventFailureCycleCounterThreshold should be configured as "127"		
Summary	The diagnostic in AUTOSAR shall provide the reporting of DTCs and related data.		
Pre-conditions	- [Diagnostic Tester] is connected to [ECU1] via TCP socket on DoIP-Port - Software components on [ECU1] are initialized. - Proxies should be available for DiagnosticOperationCycleInterface, DiagnosticConditionInterface, DiagnosticDTCInformationInterface, DiagnosticMonitorInterface and DiagnosticEventInterface		
Post-conditions	TCP connection between [Diagnostic Tester] and [ECU1] is closed.		
Main Test Execution			
Test Steps			**Pass Criteria**
Step 1	[Diagnostic Tester] Send Routing Activation Request (0x00005) with Activation type : Default(0x00).		

▽

△

Step 2	[DIAGApp01] Send Routing Activation Response.	
Step 3	[DIAGApp01] Request for service 0x85 (ControlDTCSetting) 0x85 0x02 0xFFFFFF.	[DIAGApp01] SetControlDtcStatusNotifier should be called after changing the ControlDTCSetting. [Diagnostic Manager] DM should send positive response as 0xC5 0x002.
Step 4	[DIAGApp01] Call GetControlDTCStatus.	[DIAGApp01] GetControlDTCStatus should return DTC status as kDTCSettingOff.
Step 5	[DIAGApp01] Call SetOperationCycle with "kOperationCycleStart" for "Event0".	[DIAGApp01] SetNotifier() of DiagnosticOperationCycleInterface method should be called with kOperationCycleStart. [DIAGApp01] SetDTCStatusChangedNotifier() should not be called. [DIAGApp01] SetEventStatusChangedNotifier() should not be called.
Step 6	[DIAGApp01] Call GetEventStatus.	[DIAGApp01] It should return EventStatusByte as 0x40.
Step 7	[DIAGApp01] Call GetCurrentStatus.	[DIAGApp01] It should return UdsDtcStatusBitType as 0x50.
Step 8	[DIAGApp01] Call SetCondition with "kConditionTrue" for "Event0"	[DIAGApp01] InitMonitorReason() should be called with kReenabled.
Step 9	[DIAGApp01] Call ReportMonitorAction with MonitorAction as kFailed from DiagnosticMonitor Application.	[DIAGApp01] InitMonitorReason() should not be called with MonitorAction as kFailed. [DIAGApp01] SetDTCStatusChangedNotifier() should not be called . [DIAGApp01] SetEventStatusChangedNotifier() should not be called . [DIAGApp01] SetSnapshotRecordUpdatedNotifier() should not be called for snapShotData Change for DID 1.
Step 10	[Diagnostic Tester] Call ReadDTCInformation (0x19) for reading snapShotData of DID 1 19 04 0xFF.	[DiagnosticManager] It should return previously stored DTC status and SnapShot data of DID 1.

6.2.14 [STS_DIAG_00014] Aging for UDS status bits "confirmedDTC" and "testFailedSinceLastClear"

Test Objective	The diagnostic in AUTOSAR shall provide the capability to age both the confirmedDTC bit and the testFailedSinceLastClear bit after a configurable number of aging cycles has been reached. The value at which each bit is aged may be different between the two.		
ID	STS_DIAG_00014	**State**	Draft
Affected Functional Cluster	Diagnostic		
Trace to RS Criteria	RS_Diag_04133, RS_Diag_04140		
Reference to Test Environment	STC_DIAG_00002		
Configuration Parameters	Diagnostics module: 1. DiagnosticMonitor should be configured for DiagnosticEvent "Event0" 2. DTC should be configured for the DiagnosticEvent "Event0" 3. agingAllowed should be "true" 4. DiagnosticTroubleCodeUds.udsDtcValue should be configured as "1" 5. DiagnosticEvent.eventFailureCycleCounterThreshold should be configured as "127" 6. DiagnosticAging.threshold shall be 2 7. DiagnosticAging.agingCycle shall refer to operation cycle as "POWER"		
Summary	The diagnostic in AUTOSAR shall support aging for event memory entries to remove entries from the event memory which have not failed for a specific number of operating cycles.		
Pre-conditions	- [Diagnostic Tester] is connected to [ECU1] via TCP socket on DoIP-Port - Software components on [ECU1] are initialized. - Proxies should be available for DiagnosticOperationCycleInterface, DiagnosticConditionInterface, DiagnosticDTCInformationInterface, DiagnosticMonitorInterface and DiagnosticEventInterface		
Post-conditions	TCP connection between [Diagnostic Tester] and [ECU1] is closed.		
Main Test Execution			
Test Steps			**Pass Criteria**
Step 1	[Diagnostic Tester] Send Routing Activation Request (0x00005) with Activation type : Default(0x00).		
Step 2	[DIAGApp01] Send Routing Activation Response.		
Step 3	[DIAGApp01] Call SetOperationCycle with "kOperationCycleStart" for "Event0".		[DIAGApp01] SetNotifier() of DiagnosticOperationCycleInterface method should be called with kOperationCycleStart.
Step 4	[DIAGApp01] Call SetCondition with "kConditionTrue" for "Event0"		[DIAGApp01] InitMonitorReason() should be called with kReenabled.
Step 5	[DIAGApp01] Call ReportMonitorAction with MonitorAction as kFailed from DiagnosticMonitor Application.		[DIAGApp01] InitMonitorReason() should not be called with MonitorAction as kFailed.

▽

△

Step 6	[DIAGApp01] Call SetOperationCycle with "kOperationCycleEnd" for "Event0".	[DIAGApp01] SetNotifier() of DiagnosticOperationCycleInterface method should be called with kOperationCycleEnd.
Step 7	[DIAGApp01] Call SetOperationCycle with "kOperationCycleStart" for "Event0".	[DIAGApp01] SetNotifier() of DiagnosticOperationCycleInterface method should be called with kOperationCycleStart.
Step 8	[DIAGApp01] Call ReportMonitorAction with MonitorAction as kPassed from DiagnosticMonitor Application.	[DIAGApp01] InitMonitorReason() should be called with MonitorAction as kPassed.
Step 9	[DIAGApp01] Call SetOperationCycle with "kOperationCycleStart" for "Event0".	[DIAGApp01] SetNotifier() of DiagnosticOperationCycleInterface method should be called with kOperationCycleStart.
Step 10	[DIAGApp01] Call ReportMonitorAction with MonitorAction as kPassed from DiagnosticMonitor Application.	[DIAGApp01] InitMonitorReason() should be called with MonitorAction as kPassed.
Step 11	[DIAGApp01] Call SetOperationCycle with "kOperationCycleEnd" for "Event0".	[DIAGApp01] SetNotifier() of DiagnosticOperationCycleInterface method should be called with kOperationCycleEnd.
Step 12	[Diagnostic Tester] Call ReadDTCInformation (0x19) for reading snapShotData of DID 1 19 04 0xFF.	[DiagnosticManager] It should return DTC status as 0x20.

6.2.15 [STS_DIAG_00015] Debounce counter shall be frozen, When ControlDTCSetting is set to "Disabled"

Test Objective	Testing the debounce counter behavior when ControlDTCSetting is set to "disabled".		
ID	STS_DIAG_00015	**State**	Draft
Affected Functional Cluster	Diagnostic		
Trace to RS Criteria	RS_Diag_04125		
Reference to Test Environment	STC_DIAG_00002		
Configuration Parameters	Diagnostics module: 1. DiagnosticMonitor should be configured for DiagnosticEvent "Event0" 2. DTC should be configured for the DiagnosticEvent "Event0" 3. agingAllowed should be "true" ▽		

▽

△

	△ 4. DiagnosticTroubleCodeUds.udsDtcValue should be configured as "1" 5. DiagnosticEvent.eventFailureCycleCounterThreshold should be configured as "127" 6. DiagnosticAging.threshold shall be "2" 7. DiagnosticAging.agingCycle shall refer to operation cycle as "POWER" 8. DiagEventDebounceCounterBased.counterIncrementStepSize should be "64" 9. DiagEventDebounceCounterBased.counterFailedThreshold should be "2" 10. DiagnosticDebounceAlgorithmProps.debounceBehavior should be "freeze"	
Summary	This test case covers, the debounce counter should be frozen, When ControlDTCSetting is set to "disabled".	
Pre-conditions	- [Diagnostic Tester] is connected to [ECU1] via TCP socket on DoIP-Port - Software components on [ECU1] are initialized. - Proxies should be available for DiagnosticOperationCycleInterface, DiagnosticConditionInterface, DiagnosticDTCInformationInterface, DiagnosticMonitorInterface and DiagnosticEventInterfac	
Post-conditions	TCP connection between [Diagnostic Tester] and [ECU1] is closed.	
Main Test Execution		
Test Steps		**Pass Criteria**
Step 1	[Diagnostic Tester] Send Routing Activation Request (0x00005) with Activation type : Default(0x00).	
Step 2	[DIAGApp01] Send Routing Activation Response.	
Step 3	[DIAGApp01] Call SetOperationCycle with "kOperationCycleStart" for "Event0"	[DIAGApp01] SetNotifier() of DiagnosticOperationCycleInterface method should be called with kOperationCycleStart.
Step 4	[DIAGApp01] Call SetCondition with "kConditionTrue" for "Event0"	[DIAGApp01] InitMonitorReason() should be called with kReenabled.
Step 5	[DIAGApp01] Call ReportMonitorAction with MonitorAction as kPrefailed from DiagnosticMonitor Application	
Step 6	[DIAGApp01] Call GetFaultDetectionCounter	[DIAGApp01] GetFaultDetectionCounter should return 64.
Step 7	[DIAGApp01] [Diagnostice Tester] Request for service 0x85 (ControlDTCSetting) 0x85 0x02 0xFFFFFF.	[Diagnostic Manager] DM should send positive response as 0xC5 0x002. [DIAGApp01] SetControlDtcStatusNotifier should be called after changing the ControlDTCSetting.
Step 8	[DIAGApp01] Call ReportMonitorAction with MonitorAction as kPrefailed from DiagnosticMonitor Application	
Step 9	[DIAGApp01] Call GetFaultDetectionCounter	[DIAGApp01] GetFaultDetectionCounter should return 64.

▽

△

Step 10	[DIAGApp01] Request for service 0x85 (ControlDTCSetting) 0x85 0x01 0xFFFFFF.	[DIAGApp01] SetControlDtcStatusNotifier should be called after changing the ControlDTCSetting. [Diagnostic Manager] DM should send positive response as 0xC5 0x001.
Step 11	[DIAGApp01] Call ReportMonitorAction with MonitorAction as kPrefailed from DiagnosticMonitor Application	
Step 12	[DIAGApp01] Call GetFaultDetectionCounter	[DIAGApp01] GetFaultDetectionCounter should return 127.

6.2.16 [STS_DIAG_00016] Utilization of Diagnostic service WriteDataByIdentifier (0x2E) by external Tester for receiving the Pending response (0x78) during excess payload

Test Objective	Receiving the NRC (0x78) requestCorrectlyReceivedPending response, while the write operation is been performed.		
ID	STS_DIAG_00016	**State**	Draft
Affected Functional Cluster	Diagnostic		
Trace to RS Criteria	RS_Diag_04016		
Reference to Test Environment	STC_DIAG_00001		
Configuration Parameters	- Diagnostics module: • Service instance for service WriteDataByIdentifier and ReadDataByIdentifier with DID <0x0001> are configured.		
Summary	The basic test tries to see if the tester receives an NRC(0x78) in case of excess payload during the write operation. This NRC indicates that the WriteDataByIdentifier (0x2E) request was received correctly, and that all parameters in the message are valid, but due to excess payload, the next write action to be performed is not yet completed and the server is not yet ready to receive another request.		
Pre-conditions	- [Diagnostic Tester] is connected to [ECU1] via TCP socket on DoIP-Port. - Software components on [ECU1] are initialized.		
Post-conditions	TCP connection between [Diagnostic Tester] and [ECU1] is closed.		
Main Test Execution			
Test Steps			**Pass Criteria**
Step 1	[Diagnostic Tester] Send Routing Activation Request (0x00005) with Activation type : Default(0x00)		
Step 2	[DIAGApp01] Send Routing Activation Response		

▽

△

Step 3	[Diagnostic Tester] Send UDS Request to overwrite the values <int1>: UDS-Service: WriteDataByIdentifier UDS-Payload: 0x2E ...	
Step 4	[Diagnostic Tester] Wait for invocation.	Implementation of method Write is invoked
Step 5	[Diagnostic Tester] Send UDS Request to Read the values of <int1> UDS-Service: ReadDataByIdentifier UDS-Payload: 0x22 ...	
Step 6	[Diagnostic Tester] Receive UDS response.	The negative response message with NRC (0x78) will be repeated by the server until the previous UDS requested service is completed and then the final negative or positive response is received.

6.2.17 [STS_DIAG_00017] Utilization of the UDS service RequestDownload (0x34) according to the ISO 14229-1 in manufacturer specific diagnostic session or extended diagnostic session.

Test Objective	Verification of the working of UDS services such as RequestDownload in the extended diagnostic session.		
ID	STS_DIAG_00017	**State**	Draft
Affected Functional Cluster	Diagnostic		
Trace to RS Criteria	RS_Diag_04033		
Reference to Test Environment	STC_DIAG_00001		
Configuration Parameters	- Diagnostics module: • Service instance for service RequestDownload is configured.		
Summary	This test tries to find out that following UDS service RequestDownload(0x34) according to ISO 14229-1 shall only be executed in the extended diagnostic session and should send a negative response if called for in the default session.		
Pre-conditions	- [Diagnostic Tester] is connected to [ECU1] via TCP socket on DoIP-Port. - Software components on [ECU1] are initialized.		
Post-conditions	TCP connection between [Diagnostic Tester] and [ECU1] is closed.		
Main Test Execution			
Test Steps			**Pass Criteria**
Step 1	[Diagnostic Tester] Send Routing Activation Request (0x00005) with Activation type : Default(0x00)		
Step 2	[DIAGApp01] Send Routing Activation Response		

▽

△

Step 3	[Diagnostic Tester] Send UDS Request to change content of [DIAGService01]: UDS-Service: Request download UDS-Payload: 0x34 0x01	Negative response received: Service not Supported in Active Session (0x7F 0x31 0x7F)
Step 4	[Diagnostic Tester] Send UDS request to start an Extended Diagnostic Session: UDS-Service: DiagnosticSessionControl UDS-Payload: 0x10 0x03	Positive response received (0x50 0x03).
Step 5	[Diagnostic Tester] Send UDS request to change content of [DIAGService01]: UDS-Service: Request download UDS-Payload: 0x34 0x01	
Step 6	[Diagnostic Tester] Receive UDS response.	Receive a positive response.

7 Test configuration and test steps for Logging and Tracing

7.1 Test System

7.1.1 Test configurations

Configuration ID	STC_LT_00001
Description	Standard Jenkins server for LT test
ECU 1	Hardware, 192.168.100.5
ECU 2	Hardware, 192.168.100.2
Jenkins	Jenkins Server, 192.168.100.10

The Jenkins Server, running the job with the LT Tester, is connected via Ethernet to [ECU1] hosting the System Test Application [LTApp01] and [ECU2] hosting the System Test Application [LTApp02]. The LT Tester opens TCP connections on port 3490 and receives log messages from the LT module.

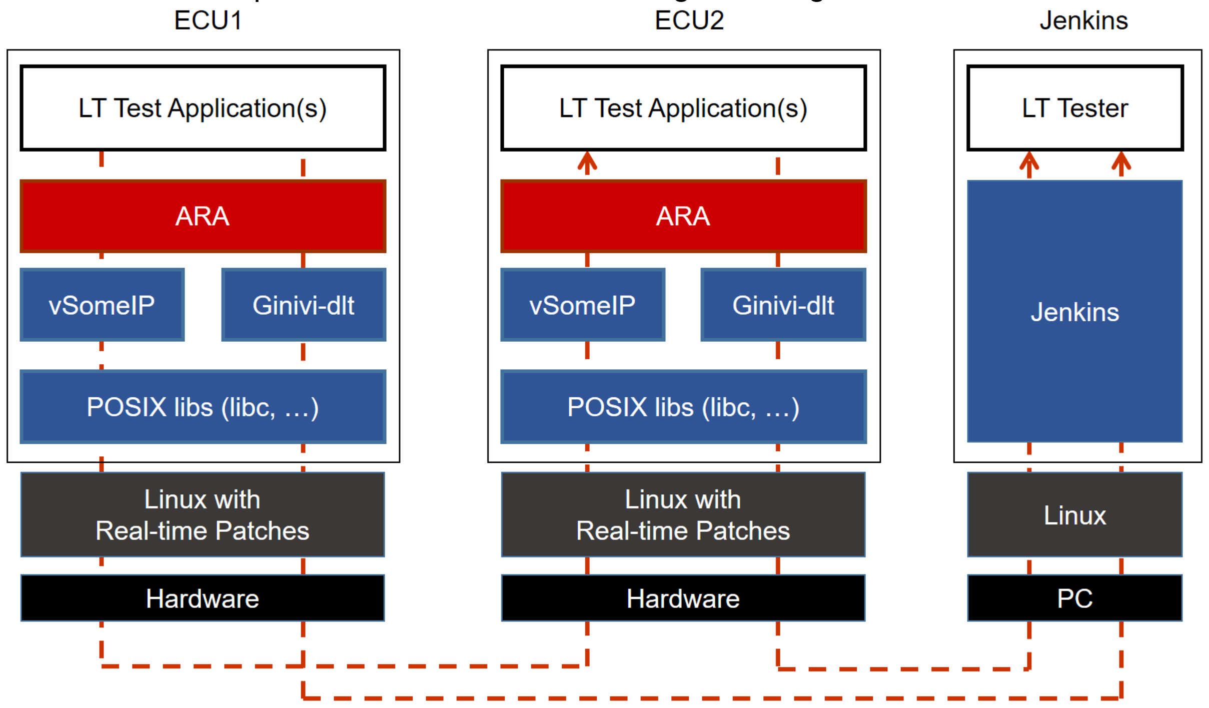

Figure 7.1: Illustration of test setup for Logging and Tracing.

7.2 Test cases

7.2.1 [STS_LT_00001] Receiving of log messages from LT module by external Tester and remote control of application's default log level.

Test Objective	Verification that all sent log messages from LT module are received by external Tester, that they carry the correct attributes like Application ID and ECU ID, and that the remote control of the application's default log level works.		
ID	STS_LT_00001	**State**	Draft
Affected Functional Cluster	Logging and Tracing		
Trace to RS Criteria	RS traceability will be added in next release		
Reference to Test Environment	STC_LT_00001 in Test configurations		
Configuration Parameters	- LT module in ECU1 is configured properly: - ECU ID for ECU1 is set to ECU1 - [LTApp01] has LT Application ID APPID1. - Context ID for [LTApp01] is set to CTX1		
Summary	The LT Tester has to connect to the LT module, which has to receive and forward the log messages from the Application Layer. First, log messages on all log levels with correct attributes are expected. Then the applications default log level is consecutively lowered to more restrictive values and it is checked, whether the respective log messages disappear.		
Pre-conditions	[LT Tester] is connected to [ECU1] via TCP socket on Port 3490. • Software components on [ECU1] are initialized. • Video Provider's default log level is set to Verbose.		
Post-conditions	TCP connection between [LT Tester] and [ECU1] is closed.		
Main Test Execution			
Test Steps		**Pass Criteria**	
Step 1	[LT Tester] Receive log messages with time-stamp.	Tester receives log messages every 0.5 seconds. The messages are received for all log levels in context with ID CTX1 and contain ECU ID ECU1, and Application ID APPID1.	
Step 2	[LT Tester] Send request to query change of [LTApp01] default log level to Debug.	Messages with log level Verbose are no longer received. Messages with lower log level are still coming in with time-stamp.	
Step 3	[LT Tester] Send request to query change of [LTApp01] default log level to Info.	Messages with log level Debug are no longer received. Messages with lower log level are still coming in with time-stamp.	
Step 4	[LT Tester] Send request to query change of [LTApp01] default log level to Warn.	Messages with log level Info are no longer received. Messages with lower log level are still coming in with time-stamp.	
Step 5	[LT Tester] Send request to query change of [LTApp01] default log level to Error.	Messages with log level Warn are no longer received. Messages with lower log level are still coming in with time-stamp.	

▽

△

Step 6	[LT Tester] Send request to query change of [LTApp01] default log level to Fatal.	Messages with log level Error are no longer received. Messages with lower log level are still coming in with time-stamp .
Step 7	[LT Tester] Send request to query change of [LTApp01] default log level to Off.	No log messages are received.

7.2.2 [STS_LT_00002] Receiving of log messages from LT modules of several ECUs.

Test Objective	Verification that all log messages from multiple ECUs are received and that they carry the correct attributes like Application ID and ECU ID.		
ID	STS_LT_00002	**State**	Draft
Affected Functional Cluster	Logging and Tracing		
Trace to RS Criteria	RS traceability will be added in next release		
Reference to Test Environment	STC_LT_00001 in Test configurations		
Configuration Parameters	- LT modules in both ECUs are configured properly. - ECU ID for [ECU1] is set to ECU1 - [LTApp01] has LT Application ID APPID1. - Context ID for [LTApp01] is set to CTX1 - ECU ID for [ECU2] is set to ECU2 - [LTApp02] has LT Application ID APPID2. - Context ID for [LTApp02] is set to CTX2		
Summary	The LT Tester has to connect to the LT modules on the different ECUs. These have to receive and forward the log messages from the different applications in the Application Layers. First, log messages from [ECU1] on all log levels with correct attributes are expected. Then a connection to [ECU2] is established and additional messages with correct attributes are expected.		
Pre-conditions	- LT Tester is connected to [ECU1] via TCP socket on Port 3490. - [LTApp01] default log level is set to Verbose. - [LTApp02] default log level is set to Verbose.		
Post-conditions	TCP connections between Jenkins server and both ECUs are closed.		
Main Test Execution			
Test Steps			**Pass Criteria**
Step 1	[LT Tester] Receive log messages.		Tester receives log messages every 0.5 seconds. The messages are received for all log levels in context with ID CTX1 and contain ECU ID ECU1, and Application ID APPID1.
Step 2	[LT Tester] Second LT Client connects to [ECU2] on Port 3490 using TCP.		Client connected.

▽

△

Step 3	[LT Tester] Receive log messages	Messages from [ECU1] are still received every 0.5 seconds. Tester additionally receives log messages from ECU2 every 0.5 seconds. The additional messages are received for log level Verbose in context with ID CTX2 and contain ECU ID ECU2, and Application ID APPID2.

7.2.3 [STS_LT_00003] Support of conversion function, get current active severity level by LT module

Test Objective	Verification that, LT module supports conversion function to get logged data in hexadecimal/binary format as a string. Verification that, LT module provides information of current severity level.		
ID	STS_LT_00003	**State**	Draft
Affected Functional Cluster	Logging and Tracing		
Trace to RS Criteria	RS traceability will be added in next release		
Reference to Test Environment	STC_LT_00004 in Test configurations		
Configuration Parameters	- LT modules on ECU1 is configured properly. - ECU ID for [ECU1] is set to ECU1 - [LTApp01] has LT Application ID APPID1. - Context ID for [LTApp01] is set to CTX1		
Summary	LT Tester connects to ECU1 to start validation of functionalities of LT module. LT tester queries LTAPP01 to get logged data in HEX/Binary format. LTAPP01 returns logged data into string with Hex/Binary representation. LT Tester queries LTAPP01 to check current log severity level.		
Pre-conditions	- LT Tester is connected to [ECU1] via TCP socket on Port 3490. - [LTApp01] default log level is set to Verbose.		
Post-conditions	TCP connections between Jenkins server and both ECUs are closed.		
Main Test Execution			
Test Steps			**Pass Criteria**
Step 1	[LT Tester] Send request to query change of [LTApp01] default log level to Debug/Warn/Info/Error.		Messages with log level Debug/Warn/Info/Error are received.
Step 2	[LT Tester] Send request to query LTAPP01 log data in hexadecimal format .		
Step 3	[LTAPP01] Prepare to send log data as a string in hexadecimal representation.		Log data provided as string in Hex
Step 4	[LT Tester] Send request to query current log level.		
Step 5	[LTAPP01] Send current log level as Debug/Warn/Info/Error.		Log level response as Debug/Warn/Info/Error.
Step 6	[LT Tester] Get log data in string.		Log data provided as string in Hex

▽

△

Step 7	[LT Tester] Send request to query LTAPP01 log data in binary format.	
Step 8	[LTAPP01] Prepare to send log data as a string in binary representation.	
Step 9	[LT Tester] Get log data in string.	Log data provided as string in binary.

8 Test configuration and test steps for Persistency

8.1 Test System

8.1.1 Test configurations

Configuration ID	STC_PER_00001
Description	Standard Jenkins server for Persistency test
ECU 1	Hardware, 192.168.100.5
Jenkins	Jenkins Server, 192.168.100.10

The Jenkins Server, running the job with the Persistency Tester is connected via Ethernet to ECU1 hosting the Persistency Test Application. The Persistency Tester is supposed to check the pass criteria.

The communication with the Persistency Test Application may take place over the Diagnostics functional cluster in form of diagnostic messages. The functionality of the Persistency Test Application described in the test steps may for example entirely be contained in routines that are implementation of subroutines of instances of the Diagnostic service RoutineControl. This service also provides a means to transport data from the Persistency Tester to the Persistency Test Application and vice versa.

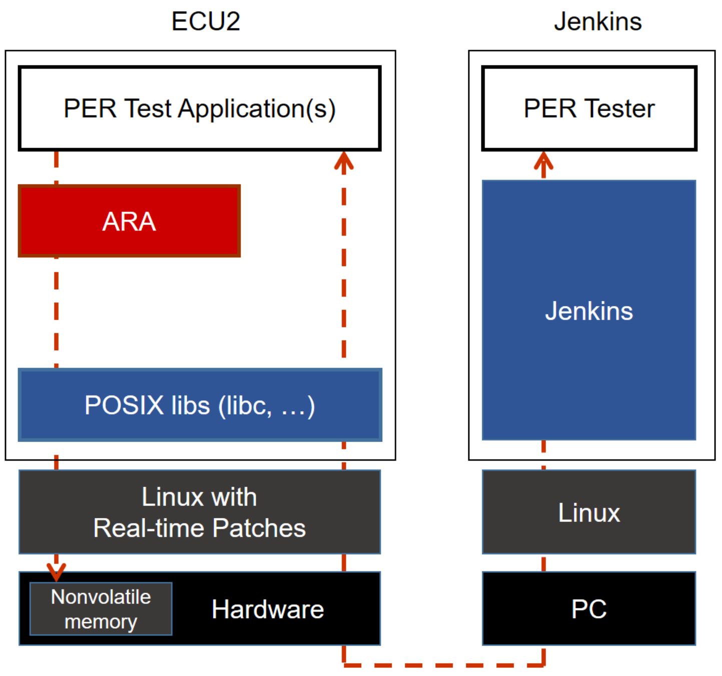

Figure 8.1: Illustration of test setup for Persistency.

8.2 Test cases

8.2.1 [STS_PER_00001] Storing an integer in a key-value database.

Test Objective	Verification, that integer data can be stored in a key-value database and that it can be retrieved again, using the associated key.		
ID	STS_PER_00001	**State**	Draft
Affected Functional Cluster	Persistency		
Trace to RS Criteria	[RS_PER_00003], [RS_PER_00010]		
Reference to Test Environment	STC_PER_00001 in Test configurations		
Configuration Parameters	- File system contains an empty file for the key-value database.		
Summary	Integer data is stored in a key-value database. It is then retrieved again from the database using the associated key and the retrieved value is compared to the original one.		
Pre-conditions	- Persistency tester is connected to ECU1. - Software components on ECU1 are initialized. - File for key-value database opened successfully and the file should be empty		
Post-conditions	TCP connection between Persistency Tester and ECU1 is closed.		
Main Test Execution			
Test Steps		**Pass Criteria**	
Step 1	[PERApp01] Store integer <intData> with associated key <intKey> in key-value database.		
Step 2	[PERApp01] Retrieve integer from key-value database using the associated key and store it in variable <retIntData>.	Originally written integer value is returned. And values of <intData> and <retInt Data> are equal.	

8.2.2 [STS_PER_00002] Storing a float in a key-value database.

Test Objective	Verification that float data can be stored in a key-value database and that it can be retrieved again, using the associated key.		
ID	STS_PER_00002	**State**	Draft
Affected Functional Cluster	Persistency		
Trace to RS Criteria	[RS_PER_00003], [RS_PER_00010]		
Reference to Test Environment	STC_PER_00001 in Test configurations		
Configuration Parameters	- File system contains an empty file for the key-value database.		

▽

△

Summary	Float data is stored in a key-value database. It is then retrieved again from the database using the associated key and the retrieved value is compared to the original one.		
Pre-conditions	- Persistency tester is connected to ECU1. - Software components on ECU1 are initialized. - File for key-value database opened successfully and the file should be empty		
Post-conditions	TCP connection between Jenkins server and ECU1 is closed.		
Main Test Execution			
Test Steps			**Pass Criteria**
Step 1	[PERApp01] Store float <floatData> with associated key <floatKey> in key-value database.		
Step 2	[PERApp01] Retrieve float from key-value database using the associated key and store it in variable <retFloatData>.		Originally written float value is returned. And Values of <floatData> and <ret FloatData> are equal

8.2.3 [STS_PER_00003] Storing a string in a key-value database.

Test Objective	Verification that string data can be stored in a key-value database and that it can be retrieved again, using the associated key.		
ID	STS_PER_00003	**State**	Draft
Affected Functional Cluster	Persistency		
Trace to RS Criteria	[RS_PER_00003], [RS_PER_00010]		
Reference to Test Environment	STC_PER_00001 in Test configurations		
Configuration Parameters	- File system contains an empty file for the key-value database.		
Summary	A string is stored in a key-value database. It is then retrieved again from the database using the associated key and the retrieved value is compared to the original one.		
Pre-conditions	- Persistency tester is connected to ECU1. - Software components on ECU1 are initialized. - File for key-value database opened successfully and the file should be empty		
Post-conditions	TCP connection between Jenkins server and ECU1 is closed.		
Main Test Execution			
Test Steps			**Pass Criteria**
Step 1	[PERApp01] Store string <stringData> with associated key <stringKey> in key-value database.		
Step 2	[PERApp01] Retrieve string from key-value database using the associated key and store it in variable <retStringData>.		Originally written string value is returned. And Values of <stringData> and <ret StringData> are equal.

8.2.4 [STS_PER_00004] Storing a string in a file.

Test Objective	Verification that a string can be stored in a file and retrieved again, using a file stream.		
ID	STS_PER_00004	**State**	Draft
Affected Functional Cluster	Persistency		
Trace to RS Criteria	[RS_PER_00004], [RS_PER_00010]		
Reference to Test Environment	STC_PER_00001 in Test configurations		
Configuration Parameters	File system contains an empty file for the file stream.		
Summary	A string is stored in a file, using a file stream. It is then retrieved again from the file and the retrieved value is compared to the original one.		
Pre-conditions	- Persistency tester is connected to ECU1. - Software components on ECU1 are initialized. - File stream successfully opened file and the file should be empty		
Post-conditions	TCP connection between Jenkins server and ECU1 is closed.		
Main Test Execution			
Test Steps		**Pass Criteria**	
Step 1	[PERApp01] Write string <stringData> to file via file stream.		
Step 2	[PERApp01] Close file.		
Step 3	[PERApp01] Open file.	File opened successfully.	
Step 4	[PERApp01] Retrieve string from file via file stream and store it in variable <retStringData>.	Originally written string value is retrieved. And Values of <stringData> and <ret StringData> are equal.	

8.2.5 [STS_PER_00005] Storing an integer in a key-value database and retrieving it after reboot.

Test Objective	Verification, that integer data can be stored in a key-value database and, after a reboot, retrieved again using the associated key.		
ID	STS_PER_00005	**State**	Draft
Affected Functional Cluster	Persistency		
Trace to RS Criteria	[RS_PER_00001], [RS_PER_00002]		
Reference to Test Environment	STC_PER_00001 in Test configurations		

▽

△

Configuration Parameters	File system contains an empty file for the key-value database.	
Summary	Integer data is stored in a key-value database. A reboot is performed and the integer data is retrieved again from the database. The retrieved value is then compared to the original one.	
Pre-conditions	- Persistency tester is connected to ECU1. - Software components on ECU1 are initialized. - File for key-value database opened successfully and the file should be empty	
Post-conditions	TCP connection between Jenkins server and ECU1 is closed.	
Main Test Execution		
Test Steps		**Pass Criteria**
Step 1	[PERApp01] Store integer <intData> with associated key <intKey> in key-value database.	
Step 2	[Persistency Tester] Request reboot.	
Step 3	[Persistency Tester] Wait until ECU1 has rebooted and PERApp01 is initialized.	
Step 4	[PERApp01] Open database.	Database file is opened.
Step 5	[PERApp01] Retrieve integer from key-value database using the associated key and store it in variable <retIntData>.	Originally written integer value is returned. And Values of <intData> and <retInt Data> are equal.

8.2.6 [STS_PER_00006] Storing a string in a file and retrieving it after reboot.

Test Objective	Verification, that string data can be stored in a file and, after a reboot, retrieved again using a file stream.		
ID	STS_PER_00006	**State**	Draft
Affected Functional Cluster	Persistency		
Trace to RS Criteria	[RS_PER_00001], [RS_PER_00002], [RS_PER_00004]		
Reference to Test Environment	STC_PER_00001 in Test configurations		
Configuration Parameters	File system contains an empty file for the file stream.		
Summary	String data is stored in a file using a file stream provided by the Persistency Functional Cluster. A reboot is performed and the string data is retrieved again from the file. The retrieved value is then compared to the original one.		
Pre-conditions	- Persistency tester is connected to ECU1. - Software components on ECU1 are initialized. - File stream successfully opened file and the file should be empty		
Post-conditions	TCP connection between Jenkins server and ECU1 is closed.		
Main Test Execution			

▽

△

Test Steps		Pass Criteria
Step 1	[PERApp01] Write string <stringData> to file via file stream.	
Step 2	[PERApp01] Close file.	
Step 3	[Persistency Tester] Request reboot.	
Step 4	[Persistency Tester] Wait until ECU1 has rebooted and PERApp01 is initialized.	
Step 5	[PERApp01] Open file.	File opened successfully.
Step 6	[PERApp01] Retrieve string from file via file stream and store it in variable <retStringData>.	Originally written string value is retrieved. And Values of <stringData> and <ret StringData> are equal.

8.2.7 [STS_PER_00007] Exceeding the maximum allowed limit for storage

Test Objective	Verification that application can't exceed the maximum limit assigned to it in persistent storage. And Testing the reporting of used storage to the application.		
ID	STS_PER_00007	**State**	Draft
Affected Functional Cluster	Persistency		
Trace to RS Criteria	[RS_PER_00011], [RS_PER_00017]		
Reference to Test Environment	STC_PER_00001 in Test configurations		
Configuration Parameters	- File system contains an empty file for the key-value database. - A configured max storage limit (Persistency-Deployment.maximumAllowedSize) for the application of size <intMaxLimit>. Limit is to be chosen as multiple of integer size (for simplicity).		
Summary	Integer data is stored as multiple copies in a key-value database using a loop. At one step, the stored copies shall exceed the maximum allowed limit of storage for the application. This last storage request shall be denied by Persistency cluster.		
Pre-conditions	- Persistency tester is connected to ECU1. - Software components on ECU1 are initialized. - File for key-value database opened successfully and the file should be empty		
Post-conditions	TCP connection between Persistency Tester and ECU1 is closed.		
Main Test Execution			
Test Steps			**Pass Criteria**
Step 1	[PERApp01] Using a loop, store multiple copies of integer <intData> with associated key <intKey> in key-value database, till reaching the maximum allowed limit <int MaxLimit>		All storage requests are accepted with no errors.

▽

△

Step 2	[PERApp01] Inside the loop, keep polling on the used storage of the key-value database. Interface to use: ara::per::GetCurrentKeyValueStorageSize (ara::core::InstanceSpecifier kvs)	The reported used storage shall be increasing till reaching the maximum allowed limit <intMaxLimit>
Step 3	[PERApp01] After the loop, Try to store another integer in the same database.	Storage request is denied.

8.2.8 [STS_PER_00008] Storing and retrieving a string in an encrypted file

Test Objective	Verification that a string can be encrypted and stored in a file and decrypted again while retrieving it from the file.		
ID	STS_PER_00008	**State**	Draft
Affected Functional Cluster	Persistency		
Trace to RS Criteria	[RS_PER_00005]		
Reference to Test Environment	STC_PER_00001 in Test configurations		
Configuration Parameters	File system contains an empty file for the file stream. CryptoJob and CryptoNeed are configured referencing any arbitary Encryption/Decryption algorithm.		
Summary	A string is stored in a file, using a file stream, in an encrypted form. It is then retrieved again from the file and decrypted. The decrypted value is compared to the original one.		
Pre-conditions	- Persistency tester is connected to ECU1. - Software components on ECU1 are initialized. - File stream successfully opened file and the file should be empty		
Post-conditions	TCP connection between Jenkins server and ECU1 is closed.		
Main Test Execution			
Test Steps		**Pass Criteria**	
Step 1	[PERApp01] Write string <stringData> to file via file stream, using the configured job of secured storage.		
Step 2	[PERApp01] Close file.		
Step 3	[PERApp01] Open file.	File opened successfully.	
Step 4	[PERApp01] Retrieve string from file via file stream and store it in variable <retStringData>.	Originally written string value is retrieved. And Values of <stringData> (before it is encrypted) and <retStringData> (after it is decrypted) are equal.	

9 Test configuration and test steps for Identity and Access Management

9.1 Test System

Identity and Access Management (IAM) requires each component to implement Policy Enforcement Point (PEP), which shall contact IAM to check access authorization of the requesting application.

System Test specification targets to check the PEP for Communication Management (FT-CM).

9.1.1 Test configurations

Configuration ID	STC_IAM_00001
Description	Standard Jenkins server for Identity and Access Management test
ECU 1	Hardware, 192.168.100.5
ECU 2	Hardware, 192.168.100.2
Jenkins	Jenkins Server, 192.168.100.10

The Jenkins Server, running the job with the IAM Tester is connected via Ethernet to [ECU1] hosting the IAM Test Application (ITA).

The IAM Tester is supposed to check the pass criteria.

The communication with the ITA may take place over the Diagnostics functional cluster in form of diagnostic messages.

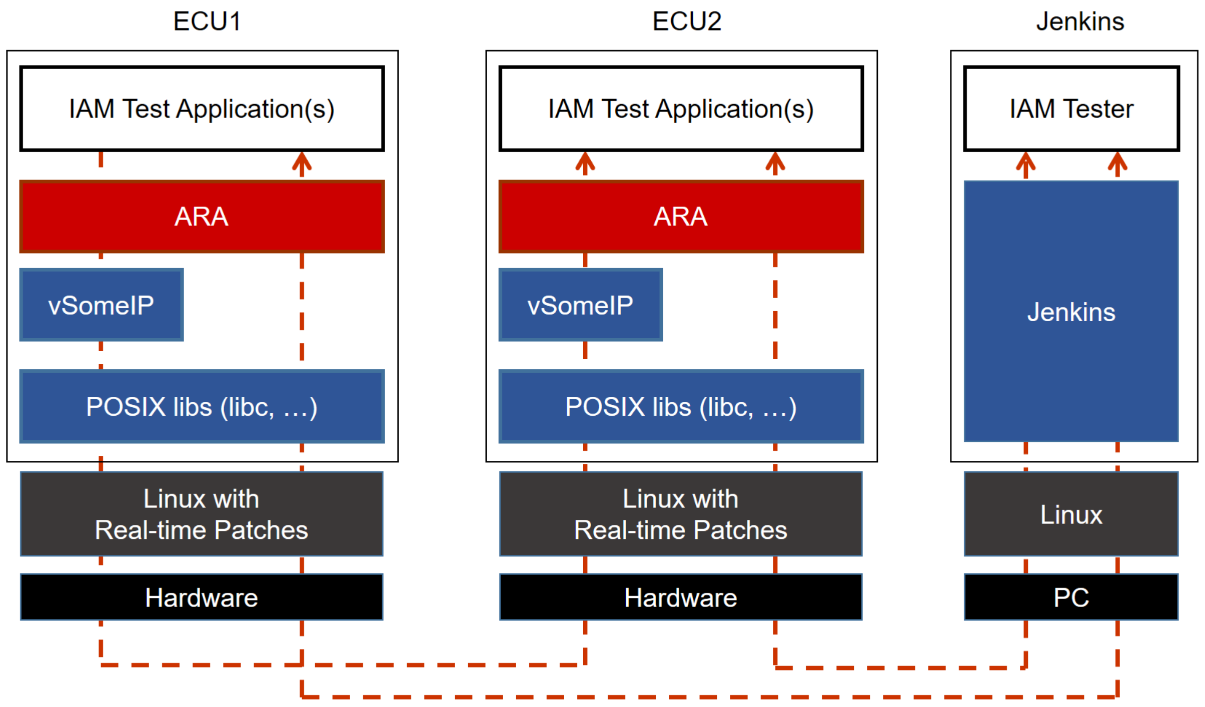

Figure 9.1: Illustration of test setup for Identity and Access Management.

9.2 Test cases

9.2.1 [STS_IAM_00001] Rejecting local service usage by an unauthorized application

Test Objective	Verification that unauthorized applications are not allowed to use services offered by another application.		
ID	STS_IAM_00001	**State**	Draft
Affected Functional Cluster	Identity and Access Management		
Trace to RS Criteria	[RS_IAM_00001], [RS_IAM_00002], [RS_IAM_00007], [RS_IAM_00010]		
Reference to Test Environment	STC_IAM_00001 in Test configurations		
Configuration Parameters	- [IAMApp01] offers and registers [IAMService01], [IAMService02], and [IAMService03] - [IAMApp02] is authorized to use [IAMService02] but not [IAMService01] and [IAMService03] - [IAMApp03] is authorized to use [IAMService03] but not [IAMService01] and [IAMService02]		
Summary	- [IAMApp02] can successfully use [IAMService02] but fails to use [IAMService01] and [IAMService03] - [IAMApp03] can successfully use [IAMService03] but fails to use [IAMService01] and [IAMService02]		
Pre-conditions	- IAM Tester is connected to [ECU1] - Software components on [ECU1] are initialized. - [ECU1] is in Machine State Parking.		

▽

△

Post-conditions	TCP connections between IAM Tester and [ECU1] is closed.	
Main Test Execution		
Test Steps		**Pass Criteria**
Step 1	[IAMApp01] Offers service [IAMService01]	
Step 2	[IAMApp01] Offers service [IAMService02]	
Step 3	[IAMApp01] Offers service [IAMService03]	
Step 4	[IAMApp02] Requests service [IAMService02]	Service discovery callback with a handle for [IAMService02] is received by [IAMApp02].
Step 5	[IAMApp03] Requests service [IAMService03]	Service discovery callback with a handle for [IAMService03] is received by [IAMApp03].
Step 6	[IAMApp02] Requests service [IAMService01]	Service is not available.
Step 7	[IAMApp02] Requests service [IAMService03]	Service is not available.
Step 8	[IAMApp03] Requests service [IAMService01]	Service is not available.
Step 9	[IAMApp03] Requests service [IAMService02]	Service is not available.

9.2.2 [STS_IAM_00002] Rejecting events sent by an unauthorized application

Test Objective	Verification that unauthorized applications are not allowed to send events.		
ID	STS_IAM_00002	**State**	Draft
Affected Functional Cluster	Identity and Access Management		
Trace to RS Criteria	[RS_IAM_00002], [RS_IAM_00007]		
Reference to Test Environment	STC_IAM_00001 in Test configurations		
Configuration Parameters	- [IAMApp01] offers and registers [IAMService01] and is authorized to send [Event11] and [Event12] - [IAMApp02] offers and registers [IAMService02] and is authorized to send [Event21] but not [Event22] - [IAMApp03] is authorized to subscribe for [Event11] and [Event21]		
Summary	- [IAMApp01] can successfully send [Event11] and [Event12] - [IAMApp02] can successfully send [Event21] but fails to send [Event22] - [IAMApp03] can successfully receive [Event11] from [IAMApp01] and [Event21] from [IAMApp02] - [IAMApp03] fails to receive [Event12] from [IAMApp01] and [Event22] from [IAMApp02]		

▽

△

Pre-conditions	- IAM Tester is connected to [ECU1] - Software components on [ECU1] are initialized. - [ECU1] is in Machine State Parking or Driving.	
Post-conditions	TCP connections between IAM Tester and [ECU1] is closed.	
Main Test Execution		
Test Steps		**Pass Criteria**
Step 1	[IAMApp01] Offers service [IAMService01] with [Event11] and [Event12]	
Step 2	[IAMApp02] Offers service [IAMService02] with [Event21]	
Step 3	[IAMApp03] Subscribes for [Event11]	Subscription is successful.
Step 4	[IAMApp03] Subscribes for [Event21]	Subscription is successful.
Step 5	[IAMApp01] Sends [Event11]	[IAMApp03] receives notification for [Event11]
Step 6	[IAMApp02] Sends [Event22]	Event is dropped silently. [IAMApp02] is not notified.
Step 7	[IAMApp02] Sends [Event21]	[IAMApp03] receives notification for [Event21]
Step 8	[IAMApp01] Sends [Event12]	[IAMApp03] does not receive notification for [Event12]

9.2.3 [STS_IAM_00003] Rejecting events if no application is authorized to receive them

Test Objective	Verification that unauthorized applications are not allowed to receive events.		
ID	STS_IAM_00003	**State**	Draft
Affected Functional Cluster	Identity and Access Management		
Trace to RS Criteria	[RS_IAM_00002], [RS_IAM_00007]		
Reference to Test Environment	STC_IAM_00001 in Test configurations		
Configuration Parameters	- [IAMApp01] offers and registers [IAMService01] and is authorized to send [Event11] and [Event12] - [IAMApp02] offers and registers [IAMService02] and is authorized to send [Event21] but not [Event22] - [IAMApp03] is authorized to receive [Event11]		
Summary	- [IAMApp01] can successfully send [Event11] and [Event12] - [IAMApp02] can successfully send [Event21] but fails to send [Event22] - [IAMApp03] can successfully receive [Event11] from [IAMApp01] - [IAMApp03] fails to subscribe for [Event12], [Event21] and [Event22]		

▽

△

Pre-conditions	- IAM Tester is connected to [ECU1] - Software components on [ECU1] are initialized. - [ECU1] is in Machine State Parking or Driving.		
Post-conditions	TCP connections between IAM Tester and [ECU1] is closed.		
Main Test Execution			
Test Steps		**Pass Criteria**	
Step 1	[IAMApp01] Offers service [IAMService01] with [Event11] and [Event12]		
Step 2	[IAMApp02] Offers service [IAMService02] with [Event21]		
Step 3	[IAMApp03] Subscribes for [Event11]	Subscription is successful.	
Step 4	[IAMApp01] Sends [Event11]	[IAMApp03] receives notification for [Event11]	
Step 5	[IAMApp01] Sends [Event12]	[Event12] is dropped and [IAMApp03] does not receive notification for [Event12]	
Step 6	[IAMApp02] Sends [Event21]	[Event21] is dropped and [IAMApp03] does not receive notification for [Event21]	
Step 7	[IAMApp02] Sends [Event22]	Event is dropped silently. [IAMApp02] is not notified.	

9.2.4 [STS_IAM_00004] Adaptive application providing access control decisions

Test Objective	Verification that an interface is provided by adaptive platform to facilitate access control decisions by adaptive application.		
ID	STS_IAM_00004	**State**	Draft
Affected Functional Cluster	Identity and Access Management		
Trace to RS Criteria	[RS_IAM_00009], [RS_IAM_00010]		
Reference to Test Environment	STC_IAM_00001 in Test configurations		
Configuration Parameters	- [IAMApp01] is an OEM application implementing PDP for access control decisions for certain resources - [IAMApp02] is authorized to use resources controlled by [IAMApp01] - [IAMApp03] is NOT authorized to use resources controlled by [IAMApp01]		
Summary	- [IAMApp01] gets requests to access resources - [IAMApp02] can successfully access resources controlled by [IAMApp01] - [IAMApp03] can NOT access resources controlled by [IAMApp01]		
Pre-conditions	- IAM Tester is connected to [ECU1] - Software components on [ECU1] are initialized. - [ECU1] is in Machine State Parking or Driving.		

▽

△

Post-conditions	TCP connections between IAM Tester and [ECU1] is closed.	
Main Test Execution		
Test Steps		**Pass Criteria**
Step 1	[IAMApp01] Offers PDP for resources (e.g. memory locations related to vehicle maintenance)	[IAMApp01] is registered as PDP in the corresponding PEP (e.g. in PER function cluster)
Step 2	[IAMApp02] Send request to access resource controlled by [IAMApp01] (e.g. a memory location)	PEP in the corresponding function cluster (e.g. PER) checks with [IAMApp01] and the request is granted
Step 3	[IAMApp03] Send request to access resource controlled by [IAMApp01] (e.g. a memory location)	PEP in the corresponding function cluster (e.g. PER) checks with [IAMApp01] and the request is NOT granted

10 Test configuration and test steps for Update and Configuration Management

10.1 Test System

The Update and Configuration Management (UCM) is responsible for update / installation / uninstallation of an Adaptive Application, an Adaptive platform itself and its underlying Operating System.There could be two use cases, Diagnostic use case and Over The Air (OTA)use case. The System Test Specification checks the functionalities provided by UCM irrespective of the use cases mentioned earlier.

10.1.1 Test configurations

Configuration ID	STC_UCM_00001
Description	Standard Jenkins server for Update and Configuration Management test
ECU 1	Hardware, 192.168.100.5
ECU 2	Hardware, 192.168.100.2
Jenkins	Jenkins Server, 192.168.100.10

The Jenkins Server is running the job with the UCM Tester which is connected via Ethernet to the [ECU1] which is hosting the UCM Test Application.

The UCM Tester is supposed to check the pass criteria.

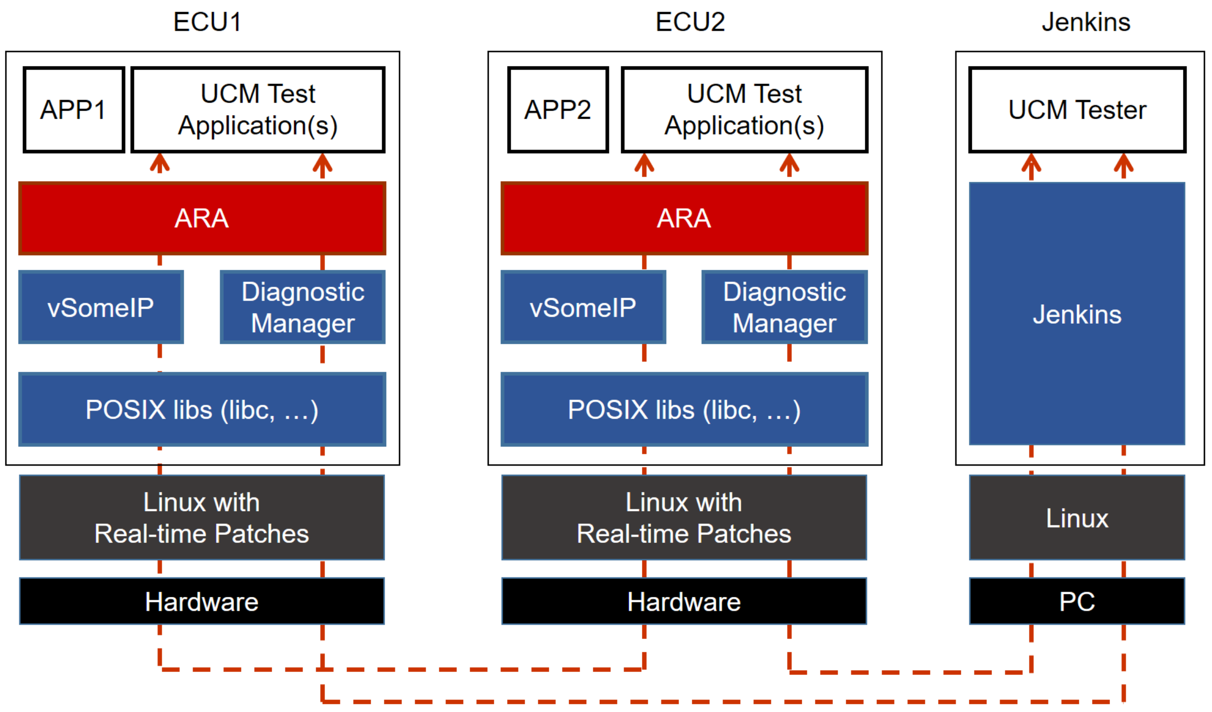

Figure 10.1: Illustration of test setup for Update and Configuration Management.

10.2 Test cases

10.2.1 [STS_UCM_00001] Check, if an update of a SW package is available.

Test Objective	Verification to check that, an Update of a SW Package is available on backend system and download the SW package, if an update is available.		
ID	STS_UCM_00001	**State**	Draft
Affected Functional Cluster	Update and Configuration Management		
Trace to RS Criteria	[RS_UCM_00010], [RS_UCM_00002], [RS_UCM_00013], [RS_UCM_00014]		
Reference to Test Environment	STC_UCM_00001 in Test configurations		
Configuration Parameters	- [UCMApp01] is configured. - [Diagnostic module] is configured.		
Summary	- UCMApp01 queries UCM to check Current SW version/name, UCMApp01 then queries to the backend system to check if any updated are available. If any updates are available, present the list of available SW packages to user. User then selects the required package and request UCMApp01 to download the requested package.		
Pre-conditions	- UCM Tester is connected to [ECU1]. - Software components on [ECU1] are initialized. - [ECU1] is in Machine State Parking.		
Post-conditions	- TCP connection between UCM Tester and [ECU1] is closed.		
Main Test Execution			
Test Steps			**Pass Criteria**
Step 1	[UCMTester]: Send a request to [UCMApp01] to read current SW version and name from UCM		
Step 2	[UCMApp01]: Start the mechanism to query read current SW version / name from UCM		
Step 3	[UCMTester]: Receive response from [UCMApp01] and store it in <UCM_SWVersion>		Payload of response contains SW version and name from UCM.
Step 4	[UCMTester]: Send a request to [UCMApp01] to read available SW version and name from Backend system		
Step 5	[UCMApp01]: Start mechanism to read all available SW Version/Name list		
Step 6	[UCMTester]: Receive response from [UCMApp01] and store it in <backend_SWVersion_List>		
Step 7	[UCMTester]: Send a request to download package <xyz> from available SW version/name list received from backend system.		
Step 8	[UCMApp01]: Start mechanism to download SW package as per specified in the request.		Requested package is downloaded successfully.

▽

△

Step 9	[UCMTester]: Send a request to read list of downloaded SW Packages	
Step 10	[UCMApp01]: Start mechanism to provide list of downloaded SW packages	Downloaded SW package list is populated successfully

10.2.2 [STS_UCM_00002] Update a SW package, on user request.

Test Objective	Verification that, a SW package is updated successfully on user request		
ID	STS_UCM_00002	**State**	Draft
Affected Functional Cluster	Update and Configuration Management		
Trace to RS Criteria	[RS_UCM_00011], [RS_UCM_00003], [RS_UCM_00023], [RS_UCM_00017], [RS_UCM_00030], [RS_UCM_00021]		
Reference to Test Environment	STC_UCM_00001 in Test configurations		
Configuration Parameters	- [UCMApp01] is configured. - [Diagnostic module] is configured.		
Summary	- UCMApp01 intends to perform multiple SW package updates. It has multiple SW packages/Updates available with it. UCM supports atomic activation(i.e. After successful transfer of multiple SW packages ,activation of all the updates/SW packages can happen on a single command) User initiates multiple SW package updates. After successful update, UCMApp01 reads SW versions/name to verify that SW packages are updated successfully. If an update was not successful then it presents Failure to user.		
Pre-conditions	- UCM Tester is connected to [ECU1]. - Software components on [ECU1] are initialized. - [ECU1] is in Machine State Parking. - SW Package is downloaded and available locally to be updated.		
Post-conditions	- TCP connection between UCM Tester and [ECU1] is closed.		
Main Test Execution			
Test Steps			**Pass Criteria**
Step 1	[UCMTester]: Send request to check availability of resources for data transfer.		
Step 2	[UCMApp01]: Start mechanism to check availability of resources.		If result == success
Step 3	[UCMTester]: Send request(Trigger from user) to update a SW package		
Step 4	[UCMApp01]: Starts mechanism to initialize it for approval.		Send an ACK message after successful initialization for performing an update.
Step 5	[UCMTester]: Send request (user approval) to update a SW package as per Package manifest (SW Version and name)		
Step 6	[UCMApp01]: Start mechanism to update a SW package.		

▽

△

Step 7	[UCMTester]: Send a request to read progress status of an update.	ACK from UCM after successful update of SW package
Step 8	[UCMApp01]: Start mechanism to provide progress status of an update of SW package.	
Step 9	[UCMTester]: Receive response of successful update of the package.	
Step 10	[UCMTester]: Send request to get SW Cluster information	
Step 11	[UCMApp01]: Start mechanism to provide SW Cluster information.	
Step 12	[UCMTester]: Receive response for SW Cluster information.	SW Cluster information should be equal to the SW Cluster package that was requested to be updated.
Step 13	Repeat Steps 1 to 12, to update another SW package.	
Step 14	[UCMTester]: Send request to Activate updated packages.	
Step 15	[UCMApp01]: Start mechanism to check SW Package dependencies.	
Step 16	[UCMTester]: Receive response of successful Activation	
Step 17	[UCMApp01]: Read value of Persistent data associated with the SW package.	Persistent data is updated in kvs database by UCM as expected.
Step 18	[UCMTester]: Send request (user approval)to update a SW package as per Package manifest (SW version and name)	
Step 19	[UCMApp01]: Start mechanism to update a SW package	
Step 20	[UCMTester]: Send request to read progress status of an Update.	
Step 21	[UCMTester]: Start mechanism to provide progress status of an update of the SW package	
Step 22	[UCMTester]: Receive response of unsuccessful update of the SW package.	
Step 23	[UCMTester]: Read value of Persistent data associated with the SW package.	Persistent data is not updated in KVS database by UCM

10.2.3 [STS_UCM_00003] Installing a SW package on user approval.

Test Objective	Verification that, a SW package is installed successfully on user request.		
ID	STS_UCM_00003	**State**	Draft
Affected Functional Cluster	Update and Configuration Management		
Trace to RS Criteria	[RS_UCM_00011], [RS_UCM_00001], [RS_UCM_00013], [RS_UCM_00017]		
Reference to Test Environment	STC_UCM_00001 in Test configurations		
Configuration Parameters	- [UCMApp01] is configured. - [Diagnostic module] is configured.		
Summary	UCMApp01 has the SW package available which is to be installed. UCMTester sends user approval for installation of a SW package to UCMApp01. UCMApp01 then queries UCM to perform SW package installation.		
Pre-conditions	- UCM Tester is connected to [ECU1]. - Software components on [ECU1] are initialized. - [ECU1] is in Machine State Parking.		
Post-conditions	- TCP connection between UCM Tester and [ECU1] is closed.		
Main Test Execution			
Test Steps		**Pass Criteria**	
Step 1	[UCMTester]: Send request to check availability of resources for data transfer		
Step 2	[UCMApp01]: Start mechanism to check availability of resources and return Result based on availability of resource.	Result == success	
Step 3	[UCMTester]: Send request (user approval) to install a SW package as per Package manifest (SW Version/name).		
Step 4	[UCMApp01]: Start mechanism to install a SW package and write/Store Persistent data associated with the SW package.		
Step 5	[UCMTester]: Response of successful installation of package	ACK from UCM after successful installation of SW package	
Step 6	[UCMTester]: Send request to read current SW version/name	SW version/name received as response should be equal to the requested SW version to be installed.	
Step 7	[UCMApp01]: Read Persistent data associated with the installed SW package from KVS database	Persistent data read is as expected .	

10.2.4 [STS_UCM_00004] Uninstalling a SW package, on user request.

Test Objective	Verification that, a SW package is uninstalled successfully on user request.		
ID	STS_UCM_00004	**State**	Draft
Affected Functional Cluster	Update and Configuration Management		
Trace to RS Criteria	[RS_UCM_00004], [RS_UCM_00005], [RS_UCM_00018]		
Reference to Test Environment	STC_UCM_00001 in Test configurations		
Configuration Parameters	- [UCMApp01] is configured. - [Diagnostic module] is configured.		
Summary	UCMApp01 has the information about the SW package to be uninstalled. UCMTester sends user approval for uninstallation of a SW package to UCMApp01. UCMApp01 then queries UCM to perform SW package uninstallation.		
Pre-conditions	- UCM Tester is connected to [ECU1]. - Software components on [ECU1] are initialized. - [ECU1] is in Machine State Parking.		
Post-conditions	- TCP connection between UCM Tester and [ECU1] is closed.		
Main Test Execution			
Test Steps			**Pass Criteria**
Step 1	[UCMTester]: Send request (Trigger from user) to uninstall a SW package and Persistent data associated with the SW package as per Package manifest.		
Step 2	[UCMApp01]: Start mechanism to uninstall a SW package.		
Step 3	[UCMTester]: Response of successful uninstallation of package		ACK from UCM after successful uninstallation of SW package
Step 4	[UCMTester]: Send request (Trigger from user) to uninstall a SW package as per package manifest		
Step 5	[UCMApp01]: Start mechanism to uninstall a SW package		
Step 6	[UCMTester]: Response of unsuccessful installation of package		NACK from UCM after unsuccessful installation of SW package
Step 7	[UCMApp01]: Read Persistent data associated with the uninstalled SW package		Persistent data should be deleted / not available

10.2.5 [STS_UCM_00005] Rollback to previous version, after corrupted SW package installation.

Test Objective	Verification that, a SW package is rolled back to its previous version after corrupted SW package installation on an adaptive Platform		
ID	STS_UCM_00005	**State**	Draft
Affected Functional Cluster	Update and Configuration Management		
Trace to RS Criteria	[RS_UCM_00008], [RS_UCM_00001], [RS_UCM_00023]		
Reference to Test Environment	STC_UCM_00001 in Test configurations		
Configuration Parameters	- [UCMApp01] is configured. - [Diagnostic module] is configured.		
Summary	- UCMTester queries UCMApp01 to update a SW package .Update of SW package fails.UCM informs UCMApp01 about the corruption. UCMApp01 then queries UCM to roll back to the previous working SW version.		
Pre-conditions	- UCM Tester is connected to [ECU1]. - Software components on [ECU1] are initialized. - [ECU1] is in Machine State Parking.		
Post-conditions	- TCP connection between UCM Tester and [ECU1] is closed.		
Main Test Execution			
Test Steps		**Pass Criteria**	
Step 1	[UCMTester]: Send request to install a SW package as per Package manifest.		
Step 2	[UCMApp01]: Start mechanism to install a SW package.		
Step 3	[UCMTester]: Send request to get SW package installation status.		
Step 4	[UCMApp01]: Start mechanism to get Installation status of a requested SW package.		
Step 5	[UCMTester]: Receive response of installation status.	Installation status is received as Failed	
Step 6	[UCMTester]: Send request to perform rollback to Previous SW version.		
Step 7	[UCMApp01]: Start mechanism to rollback to Previous SW version		
Step 8	[UCMTester]: Receive response of unsuccessful Rollback	NACK for unsuccessful Rollback	
Step 9	[UCMTester]: Send Request to rollback to previous SW package version.		
Step 10	[UCMApp01]: Start mechanism to rollback to previous SW package		
Step 11	[UCMTester]: Receive response of successful Rollback	ACK from UCM after successful rollback.	

10.2.6 [STS_UCM_00006] Read update history on an adaptive platform, on demand.

Test Objective	Verification that, an update history of an adaptive platform is available and can be read, on demand.		
ID	STS_UCM_00006	**State**	Draft
Affected Functional Cluster	Update and Configuration Management		
Reference to Test Environment	STC_UCM_00001 in Test configurations		
Trace to RS Criteria	[RS_UCM_00032]		
Configuration Parameters	- [UCMApp01] is configured. - [Diagnostic module] is configured.		
Summary	- UCMApp01 queries UCM to read Update history, UCM checks if update history is available or not. If available, it returns update information like last update time stamp, update on user approval/auto approved.		
Pre-conditions	- UCM Tester is connected to [ECU1]. - Software components on [ECU1] are initialized. - [ECU1] is in Machine State Parking.		
Post-conditions	- TCP connection between UCM Tester and [ECU1] is closed.		
Main Test Execution			
Test Steps			**Pass Criteria**
Step 1	[UCMTester]: Send request to read update history of an adaptive platform.		
Step 2	[UCMApp01]: Start mechanism to read Update history of the platform.		ACK from UCM
Step 3	[UCMTester]: Receive response from UCMApp01 with update history data.		Response from [UCMApp01] regarding update history is received. Update history may contain information like-Update version ,Time stamp, Previous version ,AUTO updated ,User updated etc.
Step 4	[UCMTester]: Send request to read update history of an adaptive platform.		
Step 5	[UCMApp01]: Start mechanism to read Update history of the platform.		NACK from UCM
Step 6	[UCMTester]: Receive response from UCMApp01 with no history data.		Response from [UCMApp01] regarding update history is not available.

10.2.7 [STS_UCM_00007]Data Transfer from Multiple clients,Simultaneously.

Test Objective	Verification to check that mutiple clients can perform data transfer of SW Packages ,simultaneously.		
ID	STS_UCM_00007	**State**	Draft

▽

△

Affected Functional Cluster	Update and Configuration Management	
Reference to Test Environment	STC_UCM_00001 in Test configurations	
Trace to RS Criteria	[RS_UCM_00019]	
Configuration Parameters	- [UCMApp01] is configured. - [UCMApp02] is configured. - [Diagnostic module] is configured.	
Summary	- UCMApp01 starts data transfer of SW package 1. - UCMApp02 also starts data trasfer of SW Package 2, simultaneously. - UCM allows UCMApp01 /UCMApp02 to perform data Trasnfer, simultaneously.	
Pre-conditions	- UCM Tester is connected to [ECU1]. - Software components on [ECU1] are initialized. - [ECU1] is in Machine State Parking.	
Post-conditions	- TCP connection between UCM Tester and [ECU1] is closed.	
Main Test Execution		
Test Steps		**Pass Criteria**
Step 1	[UCMTester]: Send request to UCMApp01 to transfer SW Package 1	
Step 2	[UCMApp01]: Start mechanism to prepare for accepting SW Package 1	
Step 3	[UCMTester]: Send request to UCMApp02 for data transfer of SW Package 2	
Step 4	[UCMApp02]: Start mechanism to prepare for accepting SW Package 2	
Step 5	[UCMTester]: Send a request to get information about transferred SW Package list	
Step 6	[UCMApp01/UCMApp02]: Receive response of list of SW Packages transferred to UCM	SWPackageList = SW Package 1 ,SW Package 2

10.2.8 [STS_UCM_00008]Install/Update/Removal of SW Package from multiple clients,sequentially.

Test Objective	Verification to check that mutiple clients can perform Install/Update/Removal of SW packages, sequentially.		
ID	STS_UCM_00008	**State**	Draft
Affected Functional Cluster	Update and Configuration Management		
Reference to Test Environment	STC_UCM_00001 in Test configurations		

▽

△

Trace to RS Criteria	[RS_UCM_00024], [RS_UCM_00026], [RS_UCM_00002]	
Configuration Parameters	- [UCMApp01] is configured. - [UCMApp02] is configured. - [Diagnostic module] is configured.	
Summary	- UCMApp01 queries UCM to Install/Update/Remove SW Package 1, UCMApp02 also queries UCM to Install/Update/Remove SW Package 2 ,simultaneously. - UCM rejects Install/Update/Removal request from UCMApp02. UCMApp02 has to wait untill UCMApp01 finishes Install/Update/Removal of SW package 1.	
Pre-conditions	- UCM Tester is connected to [ECU1]. - Software components on [ECU1] are initialized. - [ECU1] is in Machine State Parking.	
Post-conditions	- TCP connection between UCM Tester and [ECU1] is closed.	
Main Test Execution		
Test Steps		**Pass Criteria**
Step 1	[UCMTester]: Send request to read current SW version.	
Step 2	[UCMApp01]: Start mechanism to provide current SW version.	
Step 3	[UCMTester]: Receive response of current SW version and store it in <var1>.	
Step 4	[UCMTester]: Send a request to Install/Update/Remove SW Package 1 to UCMApp01.	
Step 5	[UCMApp01]: Start mechanism to Install/Update/Remove SW Package 1.	
Step 6	[UCMTester]: Send a request to read current SW version to UCMApp02	
Step 7	[UCMApp02]: Start mechanism to provide current SW version	
Step 8	[UCMTester]: Receive response as a SW version and store it in <var2>	
Step 9	[UCMTester]: Send a request to Install/Update/Remove SW Package 2 to UCMApp02	
Step 10	[UCMApp02]: Start mechanism to Install/Update/Remove SW package	
Step 11	[UCMTester]: Receive response as status of Install/Update/Removal	Status = Reject
Step 12	[UCMTester]: Send a request to UCMApp02 to get current status of UCM	
Step 13	[UCMApp02]: Start mechanism to provide UCM state	
Step 14	[UCMTester]: Receive response as UCM state .If State = Busy ,wait untill state changes to READY	UCMState = Busy/READY

▽

△

Step 15	[UCMTester]: Send request to UCMApp02 to Install/Update/Removal SW Package 2	
Step 16	[UCMApp02]: Start mechanism to prepare for Install/Update/Removal of SW Package 2	
Step 17	[UCMTester]: Receive response as successful Install/Update/Removal of SW Package 2	
Step 18	[UCMTester]: Send a request to read SW version	
Step 19	[UCMApp02]: Start mechanism to send SW version of newly installed SW Package	
Step 20	[UCMTester]: Receive response as SW version of newly installed SW Package	

10.2.9 [STS_UCM_00009]Cancel Install/Update operation of SW Package .

Test Objective	Verification to check that Install/Update operation from the client can be Cancelled.		
ID	STS_UCM_00009	**State**	Draft
Affected Functional Cluster	Update and Configuration Management		
Reference to Test Environment	STC_UCM_00001 in Test configurations		
Trace to RS Criteria	[RS_UCM_00020], [RS_UCM_00002], [RS_UCM_00003]		
Configuration Parameters	- [UCMApp01] is configured. - [Diagnostic module] is configured.		
Summary	- UCMApp01 queries UCM to install/Update a SW Package 2. - UCMApp01 later realises that there are some discrepancies, it issues Cancel request to cancel ongoing Install/Update of SW Package.		
Pre-conditions	- UCM Tester is connected to [ECU1]. - Software components on [ECU1] are initialized. - [ECU1] is in Machine State Parking.		
Post-conditions	- TCP connection between UCM Tester and [ECU1] is closed.		
Main Test Execution			
Test Steps			**Pass Criteria**
Step 1	[UCMTester]: Send request to read current version of the installed SW Package.		
Step 2	[UCMApp01]: Start mechanism to provide current version of SW Package.		

▽

△

Step 3	[UCMTester]: Receive response of current SW version and store it in <var1>.	
Step 4	[UCMTester]: Send a request to Install/Update SW Package 2	
Step 5	[UCMApp01]: Start mechanism to Install/Update SW Package 2	
Step 6	[UCMTester]: Send a request to cancel ongoing Install/Update of SW Package 2	
Step 7	[UCMApp01]: Prepare to cancel ongoing operation and send an ACK for successful cancellation.	
Step 8	[UCMTester]: Send a request to read SW version.	
Step 9	[UCMApp01]: Start mechanism to provide SW version.	
Step 10	[UCMTester]: Receive response of current SW version.	<var1> and <var2> are equal (New SW Package 2 Install/update is cancelled succesfully)

10.2.10 [STS_UCM_00010] Update underlying Operating System, on user request.

Test Objective	Verification that, underlying Operating System is updated successfully on user request		
ID	STS_UCM_00010	**State**	Draft
Affected Functional Cluster	Update and Configuration Management		
Trace to RS Criteria	[RS_UCM_00011], [RS_UCM_00023], [RS_UCM_00030], [RS_UCM_00029]		
Reference to Test Environment	STC_UCM_00001 in Test configurations		
Configuration Parameters	- [UCMApp01] is configured. - [Diagnostic module] is configured.		
Summary	- UCMApp01 has an Update available for underlying Operating System. User selects to update the available OS package. After successful update, UCMApp01 reads SW version/name to verify that OS package is updated successfully. If update was not successful then present Failure to user.		
Pre-conditions	- UCM Tester is connected to [ECU1]. - Software components on [ECU1] are initialized. - [ECU1] is in Machine State Parking. - OS Package is downloaded and available locally to be updated.		
Post-conditions	- TCP connection between UCM Tester and [ECU1] is closed.		
Main Test Execution			
Test Steps			**Pass Criteria**

▽

△

Step 1	[UCMTester]: Send request to check availability of resources for data transfer.	
Step 2	[UCMApp01]: Start mechanism to check availability of resources.	If result == success
Step 3	[UCMTester]: Send request(Trigger from user) to update the OS package.	
Step 4	[UCMApp01]: Start mechanism to initialize it for approval.	Send an ACK message after successful initialization for performing an update.
Step 5	[UCMTester]: Send request (user approval) to update the OS package as per Package manifest (SW Version and name)	
Step 6	[UCMApp01]: Start mechanism to update the OS package.	
Step 7	[UCMTester]: Send a request to read progress status of an update.	
Step 8	[UCMApp01]: Start mechanism to provide progress status of an update of OS package.	Current SW version/name should be equal to the SW version/name requested to be Updated
Step 9	[UCMTester]: Receive response of successful update of the OS package.	ACK from UCM after successful update of OS package
Step 10	[UCMTester]: Send request to Activate updated OS package.	
Step 11	[UCMApp01]: Start mechanism to check OS Package dependencies.	
Step 12	[UCMTester]: Receive response of successful Activation	
Step 13	[UCMTester]: Send request (user approval) to update OS package as per Package manifest (SW version and name)	
Step 14	[UCMApp01]: Start mechanism to update the OS package	
Step 15	[UCMTester]: Send request to read progress status of an Update.	
Step 16	[UCMTester]: Start mechanism to provide progress status of an update of the OS package	
Step 17	[UCMTester]: Receive response of unsuccessful update of the OS package.	

10.2.11 [STS_UCM_00011] Update Adaptive Platform's Functional Clusters, on user request.

Test Objective	Verification that, Functional Cluster is updated successfully on user request		
ID	STS_UCM_00011	**State**	Draft
Affected Functional Cluster	Update and Configuration Management		
Trace to RS Criteria	[RS_UCM_00011], [RS_UCM_00023], [RS_UCM_00030], [RS_UCM_00028]		
Reference to Test Environment	STC_UCM_00001 in Test configurations		
Configuration Parameters	- [UCMApp01] is configured. - [Diagnostic module] is configured.		
Summary	- UCMApp01 has an Update available for Functional Cluster. User selects to update the available package with Functional Cluster component. After successful update, UCMApp01 reads SW version/name to verify that SW package is updated successfully. If update was not successful then present Failure to user.		
Pre-conditions	- UCM Tester is connected to [ECU1]. - Software components on [ECU1] are initialized. - [ECU1] is in Machine State Parking. - SW Package is downloaded and available locally to be updated.		
Post-conditions	- TCP connection between UCM Tester and [ECU1] is closed.		
Main Test Execution			
Test Steps			**Pass Criteria**
Step 1	[UCMTester]: Send request to check availability of resources for data transfer.		
Step 2	[UCMApp01]: Start mechanism to check availability of resources.		If result == success
Step 3	[UCMTester]: Send request(Trigger from user) to update the SW package with Functional Cluster component.		
Step 4	[UCMApp01]: Start mechanism to initialize it for approval.		Send an ACK message after successful initialization for performing an update.
Step 5	[UCMTester]: Send request (user approval) to update the SW package as per Package manifest (SW Version and name)		
Step 6	[UCMApp01]: Start mechanism to update the SW package.		
Step 7	[UCMTester]: Send a request to read progress status of an update.		
Step 8	[UCMApp01]: Start mechanism to provide progress status of an update of SW package.		Current SW version/name should be equal to the SW version/name requested to be Updated
Step 9	[UCMTester]: Receive response of successful update of the SW package.		ACK from UCM after successful update of SW package

▽

△

Step 10	[UCMTester]: Send request to Activate updated SW package.	Current SW version/name should be equal to the SW version/name requested to be Updated
Step 11	[UCMApp01]: Start mechanism to check SW Package dependencies.	
Step 12	[UCMTester]: Receive response of successful Activation	
Step 13	[UCMTester]: Send request (user approval) to update SW package as per Package manifest (SW version and name)	
Step 14	[UCMApp01]: Start mechanism to update the SW package	
Step 15	[UCMTester]: Send request to read progress status of an Update.	
Step 16	[UCMTester]: Start mechanism to provide progress status of an update of the SW package	
Step 17	[UCMTester]: Receive response of unsuccessful update of the SW package.	

10.2.12 [STS_UCM_00012] Validate SW manifest and report invalid SW manifest if found inconsistent.

Test Objective	Verification that, SW manifest received during a SW update is consistent. If it is found to be inconsitent then it should report manifest error.		
ID	STS_UCM_00012	**State**	Draft
Affected Functional Cluster	Update and Configuration Management		
Trace to RS Criteria	[RS_UCM_00012]		
Reference to Test Environment	STC_UCM_00001 in Test configurations		
Configuration Parameters	- [UCMApp01] is configured. - [Diagnostic module] is configured.		
Summary	- Downloaded SW packages are available locally (with some discrepencies in the SW manifest). When UCM receives a command to install the SW package, UCM first checks consistency of the SW manifest. If there are discrepencies then it should report invalid manifest.		
Pre-conditions	- UCM Tester is connected to [ECU1]. - Software components on [ECU1] are initialized. - [ECU1] is in Machine State Parking. - SW Packages SW1 and SW2 is downloaded and available locally to be updated. - SW1 is a SW package with consistent manifest, SW2 is a SW package with an inconsistent manifest.		
Post-conditions	- TCP connection between UCM Tester and [ECU1] is closed.		
Main Test Execution			

▽

△

Test Steps		Pass Criteria
Step 1	[UCM Tester]: Send request to check availability of the resources for data transfer.	
Step 2	[UCMApp01]: Start mechanism to check availability of resources.	If result == success
Step 3	[UCMTester]: Send request(trigger from user) to update the SW package.	
Step 4	[UCMApp01]: Start mechanism to initialize it for approval.	Send an ACK message after successful initialization for performing an update.
Step 5	[UCMTester]: Send request (user approval) to update the SW package SW1.	
Step 6	[UCMApp01]: Start mechanism to submit the SW package SW1 to be updated to UCM.	
Step 7	[UCMTester]: Send request to get the status of the SW package update.	
Step 8	[UCMApp01]: Start mechanism to provide progress status of an update of the SW package SW1.	Current SW version/name should be equal to the SW version/name requested to be updated.
Step 9	[UCMTester]: Receive response of successful update of the SW package.	
Step 10	[UCMTester]: Send request to activate updated SW package.	
Step 11	[UCMApp01]: Start mechanism to check SW Package dependencies.	
Step 12	[UCMTester]: Receive response of successful Activation.	
Step 13	[UCMTester]: Send request (user approval) to update the SW package SW2.	
Step 14	[UCMApp01]: Start mechanism to submit the SW package SW2 to be updated to UCM.	Inconsistent manifest error is reported by UCM.
Step 15	[UCMTester]: Receive response invalid manifest and update request will be discarded.	

10.2.13 [STS_UCM_00013] Install/Update authenticated SW package.

Test Objective	Verification that, the SW package being installed/updated is from an authenticated source.		
ID	STS_UCM_00013	**State**	Draft

▽

△

Affected Functional Cluster	Update and Configuration Management	
Trace to RS Criteria	[RS_UCM_00006]	
Reference to Test Environment	STC_UCM_00001 in Test configurations	
Configuration Parameters	- [UCMApp01] is configured. - [Diagnostic module] is configured.	
Summary	- SW package to be updated/installed is available locally. If the signature of the SW package does not match then discard the operation.	
Pre-conditions	- UCM Tester is connected to [ECU1]. - Software components on [ECU1] are initialized. - [ECU1] is in Machine State Parking. - SW Package SW1 with valid signature, SW package SW2 with invalid signature are downloaded and available locally to be updated/installed.	
Post-conditions	- TCP connection between UCM Tester and [ECU1] is closed.	
Main Test Execution		
Test Steps		**Pass Criteria**
Step 1	[UCM Tester]: Send request to check availability of the resources for the data transfer.	
Step 2	[UCMApp01]: Start mechanism to check availability of the resources.	If result = = success.
Step 3	[UCMTester]: Send request to update/install the SW package SW1.	
Step 4	[UCMApp01]: Start mechanism to submit SW package SW1 to be installed/updated to UCM.	ACK from UCM of successful authentication of the SW package.
Step 5	[UCMTester]: Send a request to read progress status of an update.	
Step 6	[UCMApp01]: Start mechanism to provide status of the update/install.	ACK of successful update/install of the SW package.
Step 7	[UCMTester]: Send a request to update/install SW package SW2.	
Step 8	[UCMApp01]: Start mechanism to submit SW package SW2 to be installed/updated to UCM.	NACK for signature authentication failure.

10.2.14 [STS_UCM_00014] Check, if an update is available and syncing with backend server.

Test Objective	Verification to check that, UCM Master shall check if Update of a SW Package is available on back-end system and download the SW package, if an update is available.		
ID	STS_UCM_00014	**State**	Draft

▽

△

Affected Functional Cluster	Update and Configuration Management	
Trace to RS Criteria	[RS_UCM_00033], [RS_UCM_00036]	
Reference to Test Environment	STC_UCM_00001 in Test configurations	
Configuration Parameters	- [OTA Client] is configured. - [UCM Master] is configured. - [UCMApp01] is configured. - [Diagnostic module] is configured.	
Summary	- Back-end system queries to the UCM Master to check te the available software packages. UCM Master queries UCMAPP01 to check Current SW version/name, if any updates are available then the vehicle package and software packages are downloaded from back-end server to UCM Master.	
Pre-conditions	- UCM Tester is connected to OTA client. - OTA Client connected to UCM Master. - UCM Master is connected to all UCM. - UCM Tester is connected to [ECU1]. - [ECU1] and [ECU2] are connected. - Software components on [ECU1]and [ECU2] are initialized. - [ECU1] and [ECU2] is in Machine State Parking.	
Post-conditions	- TCP connection between UCM Tester and OTA Client is closed.	
Main Test Execution		
Test Steps		**Pass Criteria**
Step 1	[UCMMaster]: Notify CampaignState Idle to [OTA Client]	
Step 2	[OTA Client]: Notify CampaignState Idle to [UCMTester]	CampaignState Notification received by UCM tester.
Step 3	[UCMTester]: Send a request to OTA Client for current SW version and name.	
Step 4	[UCMMaster]: Notify CampaignState Syncing to [OTA Client]	
Step 5	[OTA Client]: Notify CampaignState Syncing to [UCMTester]	CampaignState Notification received by UCM tester.
Step 6	[OTA Client]: Start the mechanism to query read current SW version / name from UCM Master using GetSwClusterInfo.	
Step 7	[UCMMaster]: Start the mechanism to query read current SW version / name from UCM.	
Step 8	[UCMMaster]: Receive response from [UCM] and store it in <UCM_SWVersion>.	
Step 9	[OTA Client]: Receive list of available software packages from [UCMMaster].	

▽

△

Step 10	[UCMTester]: Receive list of available software packages from [OTA Client].	Payload of response contains SW version and name from all UCM aggregated by UCM Master.
Step 11	[UCMTester]: Compute the required software update	
Step 12	[UCMTester]: Send vehicle package and required software packages to [OTA Client].	
Step 13	[OTA Client]: Transfer vehicle package to [UCMMaster].	Downloads Software package successfully.
Step 14	[UCMMaster]: Notify CampaignState VehiclePackage Transfer to [OTA Client].	
Step 15	[OTA Client]: Notify CampaignState VehiclePackage Transfer to [UCMTester].	CampaignState Notification received by UCM tester.
Step 16	[OTA Client]: Transfer required software packages to [UCMMaster].	Downloads Software package successfully.

10.2.15 [STS_UCM_00015] Orchestrating a vehicle update.

Test Objective	Verification to check that, UCM Master shall orchestrate the update of software package downloaded from backend.		
ID	STS_UCM_00015	**State**	Draft
Affected Functional Cluster	Update and Configuration Management		
Trace to RS Criteria	[RS_UCM_00034], [RS_UCM_00035], [RS_UCM_00036], [RS_UCM_00037], [RS_UCM_00038], [RS_UCM_00042], [RS_UCM_00043]		
Reference to Test Environment	STC_UCM_00015		
Configuration Parameters	- [OTA Client] is configured. - [Vehicle State Manager] is configured. - [Driver Application] is configured. - [UCM Master] is configured. - [UCMApp01] is configured. - [Diagnostic module] is configured.		
Summary	- UCM Master parses the Vehicle package manifest and orchestrate the vehile update campaign.		

▽

△

Pre-conditions	- UCM Tester is connected to OTA client. - OTA Client connected to UCM Master. - UCM Master is connected to all UCM. - UCM Master is connected to Vehicle State Manager. - UCM Master is connected to Driver Application. - UCM Tester is connected to [ECU1]. - [ECU1] and [ECU2] are connected. - Software components on [ECU1]and [ECU2] are initialized. - [ECU1] and [ECU2] is in Machine State Parking.	
Post-conditions	- TCP connection between UCM Tester and OTA Client is closed.	
Main Test Execution		
Test Steps		**Pass Criteria**
Step 1	[UCMTester]: Transfer vehicle package to [OTA Client].	
Step 2	[OTA Client]: Transfer vehicle package to [UCMMaster].	Downloads Vehicle package successfully.
Step 3	[UCMMaster]: Notify CamapignState as APPROVAL_TRANSFER to [OTA Client].	Notification received by [OTA Client].
Step 4	[OTA Client]: Notify CamapignState as APPROVAL_TRANSFER to [UCM Tester].	Notification received by [UCM Tester]
Step 5	[UCMMaster]: Send request for safety policy.	
Step 6	[Vehicle State Manager]: Send safe to update notification.	Notification received by [UCM Master].
Step 7	[UCMMaster]: Send request for user approval for transfer.	
Step 8	[Driver Application]: Sends user approval for transfer.	Notification received by [UCM Master].
Step 9	[UCMMaster]: Notify CamapignState as TRANSFERRING to [OTA Client].	Notification received by [OTA Client].
Step 10	[OTA Client]: Notify CamapignState as TRANSFERRING to [UCM Tester].	Notification received by [UCM Tester].
Step 11	[UCMMaster]: Transfer software package to [UCM].	Downloads Vehicle package successfully in UCM.
Step 12	[UCMMaster]: Notify CamapignState as APPROVAL_PROCESSING to [OTA Client].	Notification received by [OTA Client]
Step 13	[OTA Client]: Notify CamapignState as APPROVAL_PROCESSING to [UCMTester].	Notification received by [UCM Tester].
Step 14	[UCMMaster]: Send request for safety policy.	
Step 15	[Vehicle State Manager]: Send safe to update notification.	Notification received by [UCM Master].

▽

△

Step 16	[UCMMaster]: Send request for user approval for processing.	
Step 17	[Driver Application]: Sends user approval for processing.	Notification received by [UCM Master].
Step 18	[UCMMaster]: Notify CamapignState as PROCESSING to [OTA Client].	Notification received by [OTA Client].
Step 19	[OTA Client]: Notify CamapignState as PROCESSING to [UCMTester].	Notification received by [UCM Tester].
Step 20	[UCMMaster]: Process software package to [UCM].	
Step 21	[UCMMaster]: Notify CamapignState as APPROVAL_ACTIVATE to [OTA Client].	Notification received by [OTA Client].
Step 22	[OTA Client]: Notify CamapignState as APPROVAL_ACTIVATE to [UCMTester].	Notification received by [UCM Tester].
Step 23	[UCMMaster]: Send request for safety policy.	
Step 24	[Vehicle State Manager]: Send safe to update notification.	Notification received by [UCM Master].
Step 25	[UCMMaster]: Send request for user approval for activate.	
Step 26	[Driver Application]: Sends user approval for activate.	Notification received by [UCM Master].
Step 27	[UCMMaster]: Activate software package to [UCM].	
Step 28	[UCMMaster]: Notify CamapignState as ACTIVATED to [OTA Client].	Notification received by [OTA Client].
Step 29	[OTA Client]: Notify CamapignState as ACTIVATED to [UCMTester].	Notification received by [UCM Tester]
Step 30	[UCMMaster]: finish software package to [UCM].	
Step 31	[UCMMaster]: Notify CamapignState as IDLE to [OTA Client].	Notification received by [OTA Client].
Step 32	[OTA Client]: Notify CamapignState as IDLE to [UCMTester].	Notification received by [UCM Tester].
Step 33	[OTA Client]: Gethistory request to [UCMMaster].	Activation history from [UCM master].

11 Test configuration and test steps for E2E Protection

11.1 Test System

11.1.1 Test configurations E2E Protection

Configuration ID	STC_E2E_00001
Description	Nominal AP Apps for E2E Protection
ECU 1	Hardware, 192.168.100.5
ECU 2	Hardware, 192.168.100.2
Jenkins	Jenkins Server, 192.168.100.10

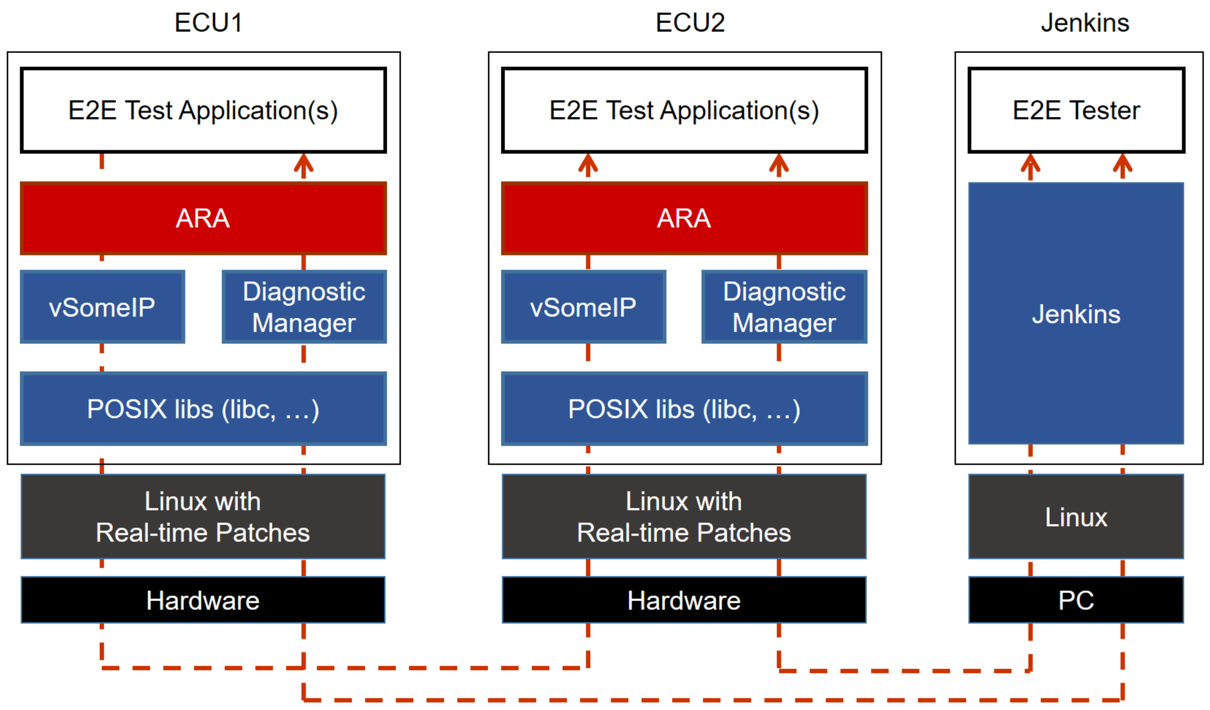

Figure 11.1: Illustration of test setup for STC-E2E-00001.

Configuration ID	STC_E2E_00002
Description	Nominal AP Apps for E2E Protection + Corrupting App Intervention
ECU 1	Hardware, 192.168.100.5
ECU 2	Hardware, 192.168.100.2
Jenkins	Jenkins Server, 192.168.100.10

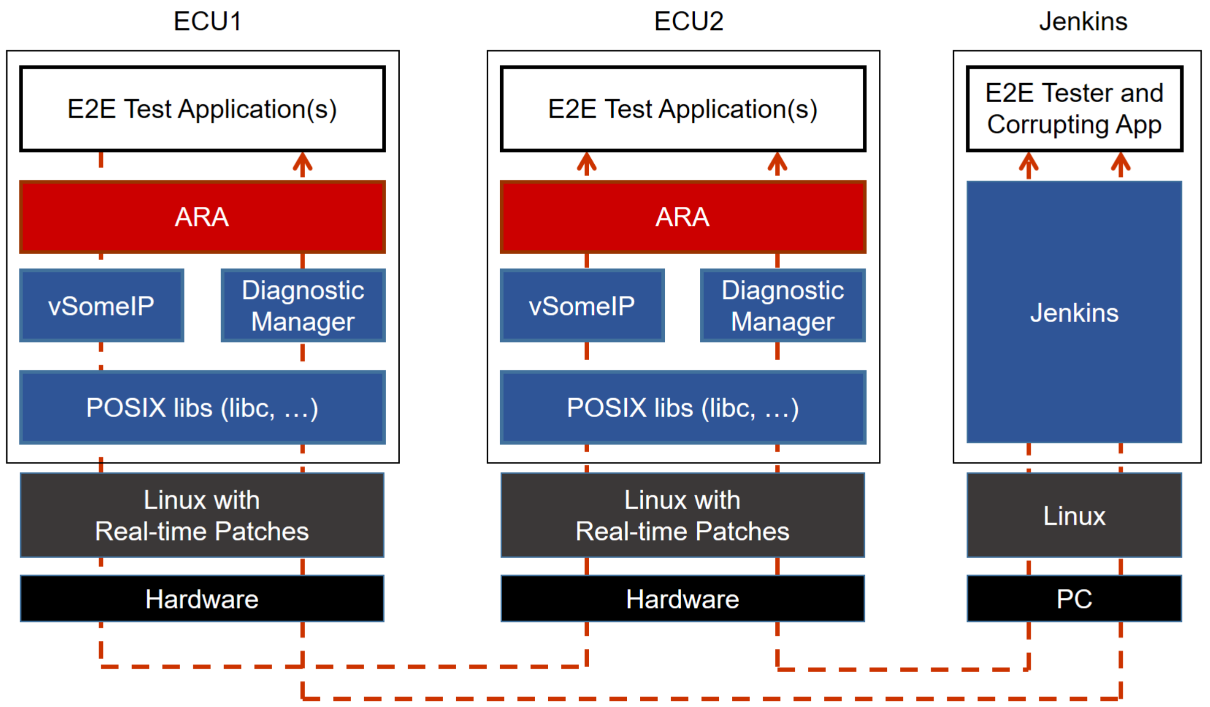

Figure 11.2: Illustration of test setup for STC-E2E-00002.

The Jenkins Server, running the job with the E2E protection test ([E2E Tester]) is connected via Ethernet to [ECU1] and [ECU2].

The [E2E Tester] is supposed to collect the results.

The communication between [E2E Tester] and the applications on ECU may take place over the Diagnostics functional cluster in form of diagnostic messages.

11.2 Test cases

11.2.1 [STS_E2E_00001] E2E Protection from AP to AP (Event Communication)

Test Objective	To verify that the E2E protection is done properly between applications in adaptive platforms		
ID	STS_E2E_00001	**State**	Draft
Affected Functional Cluster	Safety		
Trace to RS Criteria	[RS_E2E_08539], [RS_E2E_08540], [RS_E2E_08543], [RS_E2E_08544]		
Reference to Test Environment	STC_E2E_00001 in Test configurations E2E Protection		
Configuration Parameters	- Event based communication. - The existing communication services comprise the following (service & data names are arbitrary): - [E2EService01]: Offered by [E2EApp01], requested by [E2EApp02]. - <Data1> is protected by E2E, sent by [E2EApp01] and received by [E2EApp02].		

▽

△

Summary	[E2EService01] is offered by [E2EApp01] on ECU1 and is requested by [E2EApp02] on ECU2. [E2EApp01] sends <Data1> to [E2EApp02] in a certain cycle time. If it cannot be sent within a certain cycle time, E2E will detect an error.	
Pre-conditions	- [E2E Tester] is connected to both ECUs. - Both ECUs are in Machine State Parking. - [E2EApp01] and [E2EApp02] are shut down according to Machine State.	
Post-conditions	E2E Tester is disconnected to both ECUs.	
Main Test Execution		
Test Steps		**Pass Criteria**
Step 1	[E2E Tester] Request for change of Machine State to Driving from Execution Manager. Machine State for ECU1 and ECU2 are changed to Driving, and [E2EApp01] and [E2EApp02] are started up.	
Step 2	[E2EApp01] Offer service [E2EService01].	
Step 3	[E2EApp02] Request service [E2EService01].	
Step 4	[E2EApp01] Send E2E protected <Data1> with arbitrary values. The length of <Data1> is 4kbyte	
Step 5	[E2EApp02] Call GetProfileCheckStatus() for <Data1>.	[E2EApp02] reads ProfileCheckStatus = Ok
Step 6	[E2EApp02] Execute Update for <Data1>.	[E2EApp02] receives correct value of <Data1>
Step 7	Repeat setp4 to step6 for 10 times. Repeated in a certain cycle time. Every time length of <Data1> is changed.	<Data1> is always received with correct values. ProfileCheckStatus is always = OK except Step8
Step 8	[E2EApp01] Wait for more than cycle time. <Data1> is not sent once in 10 times within a certain cycle time.	[E2EApp02] reads ProfileCheckStatus = Repeated

The following sequence diagram shows the schematic operation of STS_E2E_00001. (Note that not all test steps are represented exactly.)

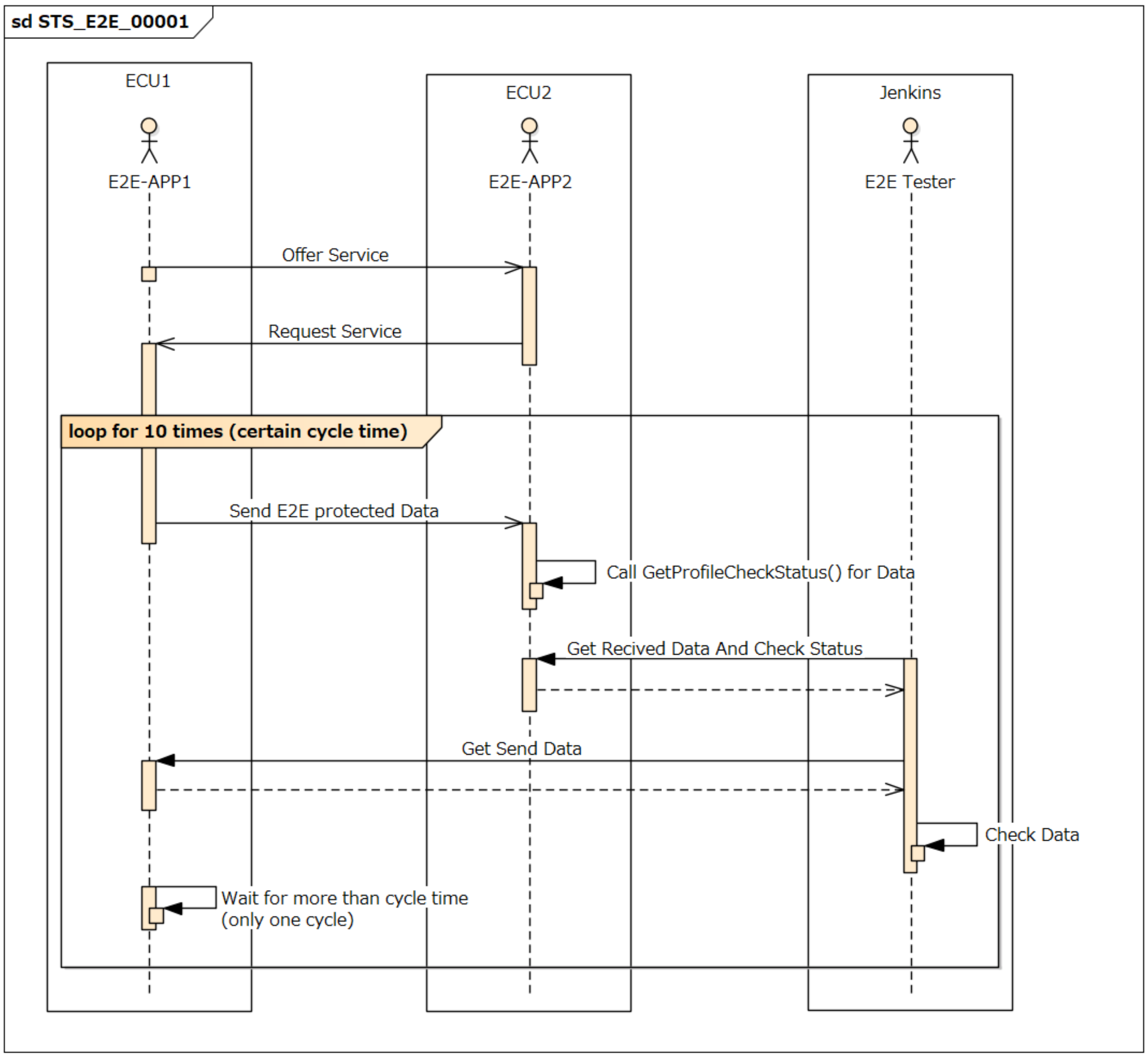

Figure 11.3: Sequence diagram of STS_E2E_00001.

11.2.2 [STS_E2E_00002] Corrupting App Affecting Communication

Test Objective	To verify that the Corrupting App to simulate a corrupted communication is detected by E2E		
ID	STS_E2E_00002	**State**	Draft
Affected Functional Cluster	Safety		
Trace to RS Criteria	[RS_E2E_08529], [RS_E2E_08534], [RS_E2E_08545], [RS_E2E_08546], [RS_E2E_08547], [RS_E2E_08548]		
Reference to Test Environment	STC_E2E_00002 in Test configurations E2E Protection		

▽

△

Configuration Parameters	- maxDeltaCounter is set to 5. - windowSizeInit is set to 2. - windowSizeValid is set to 2. - windowSizeInvalid is set to 2. - minOkStateInit is set to 1. - maxErrorStateInit is set to 1. - minOkStateValid is set to 1. - maxErrorStateValid is set to 1. - minOkStateInvalid is set to 1. - maxErrorStateInvalid is set to 1. - clearFromValidToInvalid is set to 0. - Event based communication. - The existing communication services comprise the following (service & data names are arbitrary): - [E2EService01]: Offered by [E2EApp01], requested by [E2EApp02]. - <Data1> is protected by E2E, sent by [E2EApp01] and received by [E2EApp02]. - [E2EDataCorrupter01] to send <Data1>, with similar message format as sent by [E2EApp01]	
Summary	[E2EService01] is offered by [E2EApp01] on ECU1 and is requested by [E2EApp02] on ECU2. [E2EApp01] sends <Data1> to [E2EApp02]. [E2EDataCorrupter01] sends the same communication data sent by [E2EApp01], but it has corrupted data. [E2EApp02] detects the corrupted data thanks to the E2E protection.	
Pre-conditions	- [E2E Tester] is connected to both ECUs. - Both ECUs are in Machine State Parking. - [E2EApp01] and [E2EApp02] are shut down according to Machine State.	
Post-conditions	E2E Tester is disconnected to both ECUs.	
Main Test Execution		
Test Steps		**Pass Criteria**
Step 1	[E2E Tester] Request for change of Machine State to Driving from Execution Manager. Machine State for ECU1 and ECU2 are changed to Driving, and [E2EApp01] and [E2EApp02] are started up.	
Step 2	[E2EApp01] Offer service [E2EService01].	
Step 3	[E2EApp02] Request service [E2EService01].	
Step 4	[E2EApp01] Send E2E protected <Data1> twice with arbitrary values.	
Step 5	[E2EApp02] • Call GetProfileCheckStatus() for <Data1> • Call GetE2EStateMachineState()	[E2EApp02] • reads ProfileCheckStatus = Ok • reads SMState = Valid

▽

△

Step 6	[E2EDataCorrupter01] Send the same communication data as <Data1> sent by [E2EApp01], but it has corrupted data.	[E2EApp02] • reads ProfileCheckStatus = Error (CRC error) • reads SMState = Valid
Step 7	[E2EDataCorrupter01] Send the same communication data as <Data1> sent by [E2EApp01], but it has the corrupted DataID field.	[E2EApp02] • reads ProfileCheckStatus = Error (CRC error) • reads SMState = Invalid
Step 8	[E2EApp01] Send E2E protected <Data1> with arbitrary values.	[E2EApp02] • reads ProfileCheckStatus = Ok • reads SMState = Valid
Step 9	[E2EDataCorrupter01] Send the same communication data as <Data1> sent by [E2EApp01], but it has the corrupted Counter field and the recalculated CRC field for <Data1>. (The Counter value which added maxDeltaCounter or more should be set.)	[E2EApp02] • reads ProfileCheckStatus = WrongSequence • reads SMState = Valid
Step 10	[E2EDataCorrupter01] Send the same communication data as <Data1> sent by [E2EApp01], but it has the same Counter value as last time.	[E2EApp02] • reads ProfileCheckStatus = Repeated • reads SMState = Invalid

The following sequence diagram shows the schematic operation of STS_E2E_00002. (Note that not all test steps are represented exactly.)

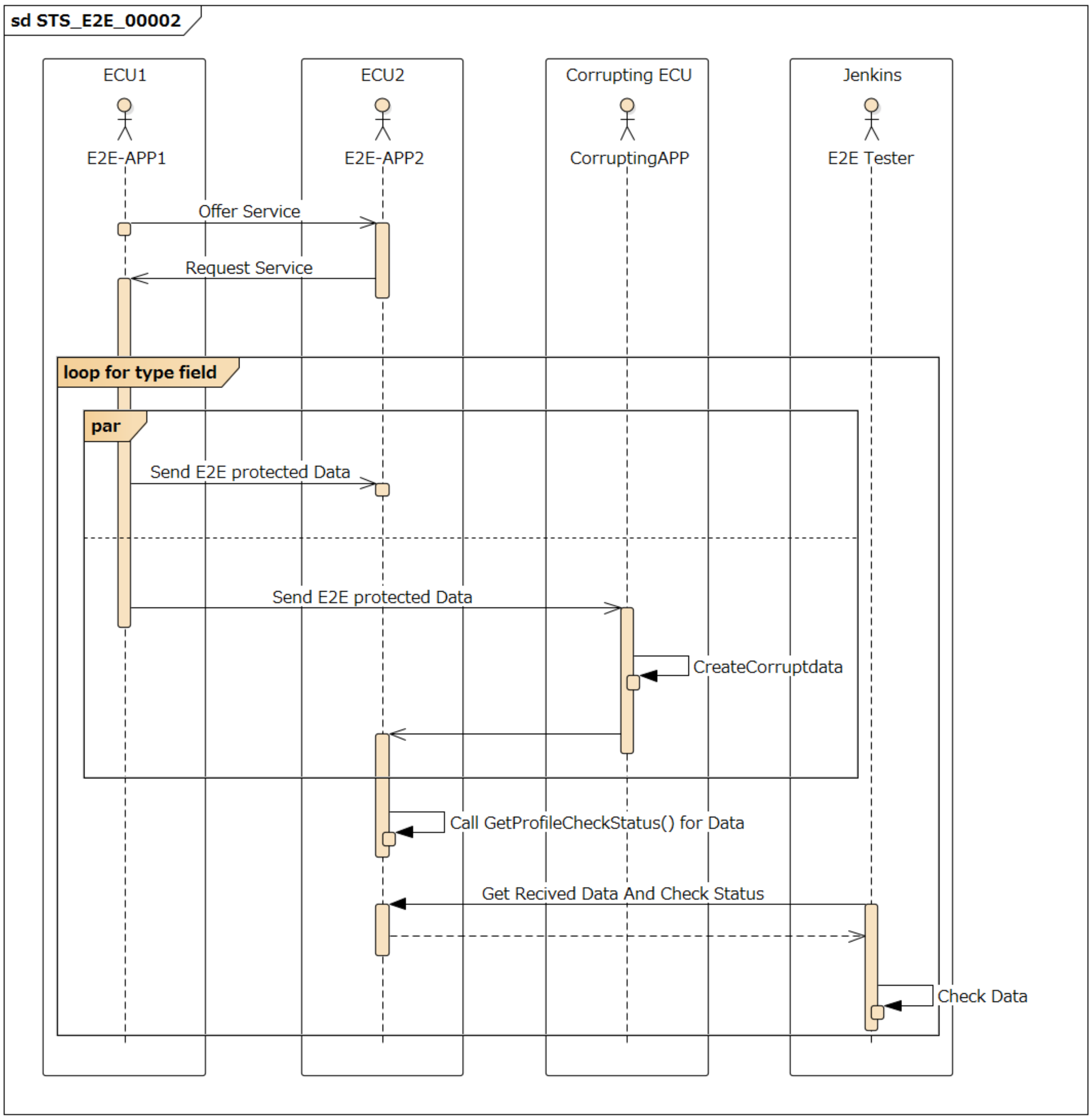

Figure 11.4: Sequence diagram of STS_E2E_00002.

11.2.3 [STS_E2E_00003] E2E Protection from AP to AP (Method Communication)

Test Objective	To verify that the E2E protection is done properly between applications in adaptive platforms		
ID	STS_E2E_00003	**State**	Draft
Affected Functional Cluster	Safety		

▽

△

Trace to RS Criteria	[RS_E2E_08541]	
Reference to Test Environment	STC_E2E_00001 in Test configurations E2E Protection	
Configuration Parameters	- Method based communication. - The existing communication services comprise the following (service & data names are arbitrary): - [E2EService01]: Offered by [E2EApp01], requested by [E2EApp02]. - [E2EService01] service receives requested services synchronously. - <Data1> is an argument to the [E2EService01].	
Summary	[E2EService01] is offered by [E2EApp01] on ECU1 and is requested by [E2EApp02] on ECU2. The [E2EApp02] on [ECU2] receives data over service [E2EService01] from [E2EApp01] as synchronous service call.	
Pre-conditions	- [E2E Tester] is connected to both ECUs. - Both ECUs are in Machine State Parking. - [E2EApp01] and [E2EApp02] are shut down according to Machine State.	
Post-conditions	E2E Tester is disconnected to both ECUs.	
Main Test Execution		
Test Steps		**Pass Criteria**
Step 1	[E2E Tester] Request for change of Machine State to Driving from Execution Manager. Machine State for ECU1 and ECU2 are changed to Driving, and [E2EApp01] and [E2EApp02] are started up.	
Step 2	[E2EApp01] Offer service [E2EService01].	
Step 3	[E2EApp02] Request service [E2EService01] with the argument <Data1>.	
Step 4	[E2EApp02] Receive data over service [E2EService01] from [E2EApp01].	[E2EApp02] Data is received from [E2EApp01] over service [E2EService01].
Step 5	[E2EApp02] Call GetProfileCheckStatus().	[E2EApp02] reads ProfileCheckStatus = Ok
Step 6	[E2EApp02] Store received data.	
Step 7	Repeat setp3 to step6 for multiple times. Every time <Data1> is changed.	ProfileCheckStatus is always = Ok [E2EApp02] always receives the correct value.

The following sequence diagram shows the schematic operation of STS_E2E_00003. (Note that not all test steps are represented exactly.)

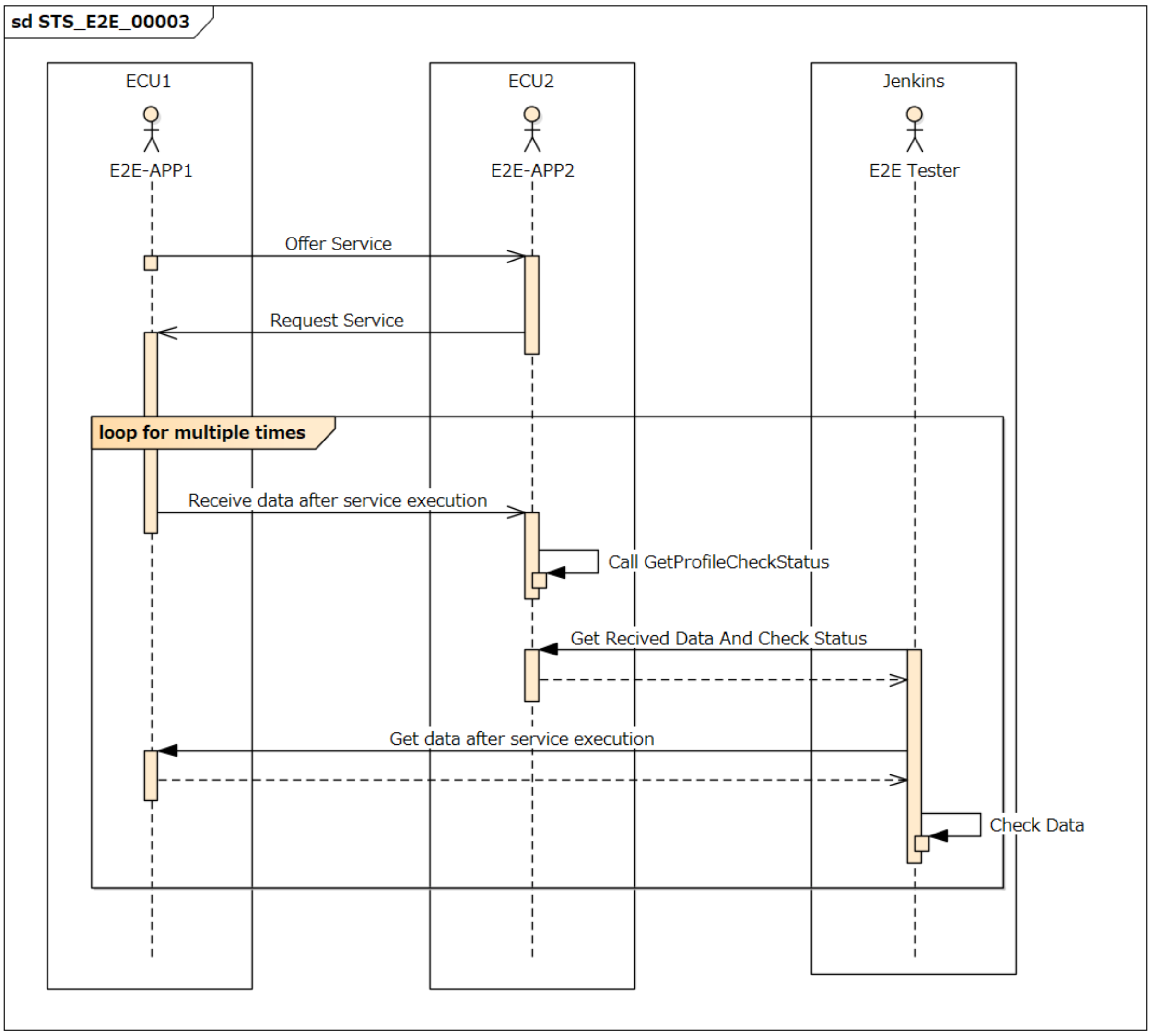
sd STS_E2E_00003
ECU1
E2E-APP1
ECU2
E2E-APP2
Jenkins
E2E Tester
Offer Service
Request Service
loop for multiple times
Receive data after service execution
Call GetProfileCheckStatus
Get Recived Data And Check Status
Get data after service execution
Check Data

Figure 11.5: Sequence diagram of STS_E2E_00003.

12 Test configuration and test steps for Time Synchronization

12.1 Test System

12.1.1 Test configurations

Configuration ID	STC_TS_00001
Description	Standard Jenkins server for Time Synchronization test
ECU 1	Hardware, 192.168.100.5
ECU 2	Hardware, 192.168.100.2
Jenkins	Jenkins Server, 192.168.100.10

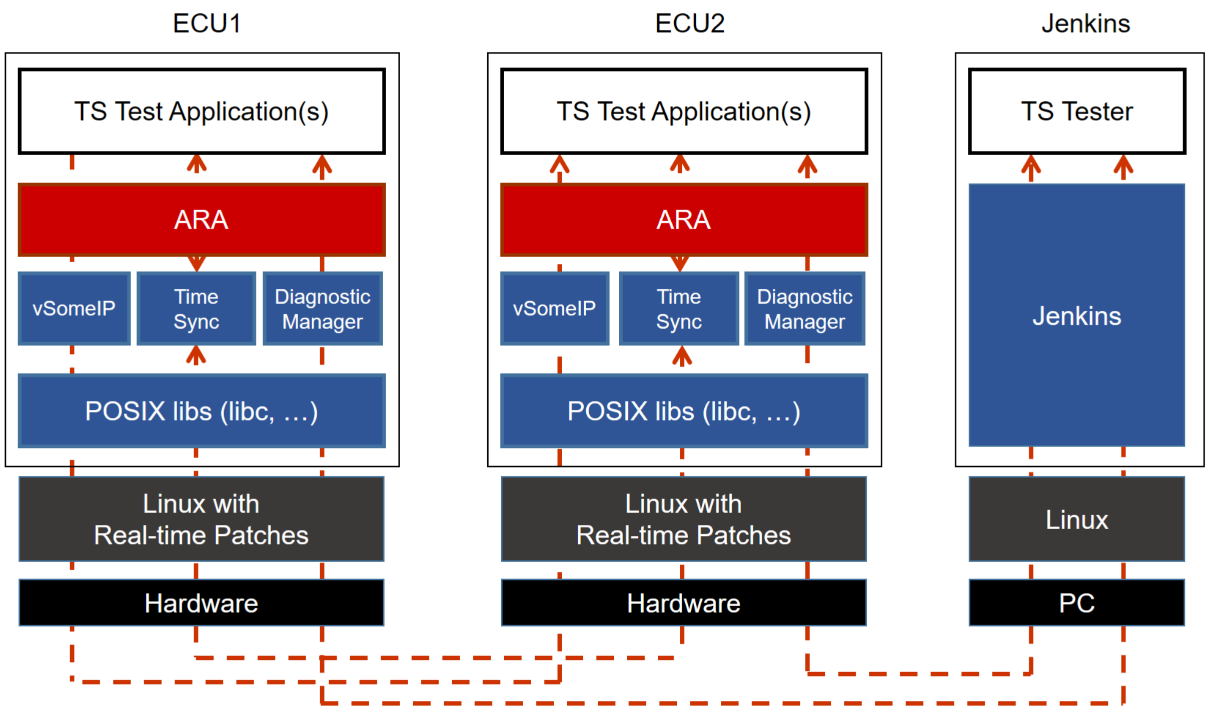

Figure 12.1: Illustration of test setup for Time Synchronization.

The Jenkins Server, running the job with the Time Synchronization test ([TS Tester]) is connected via Ethernet to [ECU1] hosting the System Test Application [TSApp01] and [ECU2] hosting the System Test Application [TSApp02].

The [TS Tester] is supposed to collect the results.

The communication between [TS Tester] and the applications on ECU may take place over the Diagnostics functional cluster in form of diagnostic messages.

12.2 Test cases

12.2.1 [STS_TS_00001] Check APIs of Offset Slave TimeBase (TB)

Test Objective	Verification that whether APIs of a Offset Slave TB can be used correctly.		
ID	STS_TS_00001	**State**	Draft
Affected Functional Cluster	Time Synchronization		
Trace to RS Criteria	[RS_TS_00005], [RS_TS_00012], [RS_TS_00013], [RS_TS_00017], [RS_TS_00021], [RS_TS_00026], [RS_TS_00030]		
Reference to Test Environment	STC_TS_00001 in Test configurations		
Configuration Parameters	- [ECU1] is synced by [ECU2]. - [ECU2] is Global Time Master. - [ECU1] has a Offset Slave TB and a Synchronized Slave TB. - [ECU2] has a Offset Master TB and a Synchronized Master TB. - The Synchronized Slave TB on [ECU1] is synced by the Synchronized Master TB on [ECU2]. - The Offset Slave TB on [ECU1] depend on the Synchronized Slave TB on [ECU1], - The Offset Master TB on [ECU2] depend on the Synchronized Mater TB on [ECU2].		
Summary	Verification that [TSApp01] can use APIs of Offset Slave TB.		
Pre-conditions	- [TS Tester] is connected to [ECU1]. - [ECU1] is in Machine State Parking. - [TSApp01] is shut down according to Machine State.		
Post-conditions	[TS Tester] is disconnected to [ECU1].		
Main Test Execution			
Test Steps		**Pass Criteria**	
Step 1	[TS Tester] Request for change of Machine State to Driving from Execution Manager. Machine State for [ECU1] is changed to Driving, and [TSApp01] is started up.		
Step 2	[TSApp01] Find the Offset Slave TB on [ECU1].	The Offset Slave TB on [ECU1] is found successfully.	
Step 3	[TSApp01] Configure the Offset Slave TB on [ECU1].		
Step 4	[TSApp01] Get rate deviation of the Offset Slave TB on [ECU1].	Rate deviation is got successfully.	
Step 5	[TSApp01] Get Time Base Status of the Offset Slave TB on [ECU1].	Time Base Status is got successfully.	
Step 6	[TSApp01] Get a getType of the Offset Slave TB on [ECU1].	The getType is Offset Slave TB.	
Step 7	[TSApp01] Set Offset value of the Offset Slave TB on [ECU1].		

▽

△

Step 8	[TSApp01] Get Offset value of the Offset Slave TB on [ECU1].	Offset value is the value set in Step 7.
Step 9	[TSApp01] Get current time of the Offset Slave TB on [ECU1].	Current time is got successfully.
Step 10	[TSApp01] Start the timer of the Offset Slave TB on [ECU1] so that the timer will expire at the specified time.	
Step 11	[TSApp01] When time-up is notified. Get current time of the Offset Slave TB on [ECU1].	Current time is the specified time.

12.2.2 [STS_TS_00002] TimeSynchronization of applications between ECUs.

Test Objective	Verification that synchronization between the application on [ECU1] and [ECU2] can correctly be done.		
ID	STS_TS_00002	**State**	Draft
Affected Functional Cluster	Time Synchronization		
Trace to RS Criteria	[RS_TS_00005], [RS_TS_00026], [RS_TS_20052], [RS_TS_20053]		
Reference to Test Environment	STC_TS_00001 in Test configurations		
Configuration Parameters	- [ECU1] is synced by [ECU2]. - [ECU2] is Global Time Master. - [ECU1] has a Offset Slave TimeBase(TB) and a Synchronized Slave TB. - [ECU2] has a Offset Master TB and a Synchronized Master TB. - The Synchronized Slave TB on [ECU1] is synced by the Synchronized Master TB on [ECU2]. - The Offset Slave TB on [ECU1] depend on the Synchronized Slave TB on [ECU1], - The Offset Master TB on [ECU2] depend on the Synchronized Mater TB on [ECU2]. - Event based communication. - The existing communication services comprise the following (service & data names are arbitrary): • [TSService01]: Offered by [TSApp01], requested by [TSApp02]. • [TSService01]: [TSApp01] send a synchronization time to [TSApp02].		
Summary	Verification that [TSApp01] and [TSApp02] can be synchronized.		
Pre-conditions	- [TS Tester] is connected to both ECUs. - Both ECUs are in Machine State Parking. - [TSApp01] and [TSApp02] are shut down according to Machine State.		
Post-conditions	[TS Tester] is disconnected to both ECUs.		
Main Test Execution			
Test Steps		**Pass Criteria**	

▽

△

Step 1	[TS Tester] Request for change of Machine State to Driving from Execution Manager. Machine State for [ECU1] and [ECU2] are changed to Driving, and [TSApp01] and [TSApp02] are started up.	
Step 2	[TSApp01] Offer service [TSService01].	
Step 3	[TSApp02] Request service [TSService01].	
Step 4	[TSApp01] Find the Offset Slave TB on [ECU1].	The Offset Slave TB on [ECU1] is found successfully.
Step 5	[TSApp01] Configure the Offset Slave TB on [ECU1].	
Step 6	[TSApp02] Find the Offset Master TB on [ECU2].	The Offset Master TB on [ECU2] is found successfully.
Step 7	[TSApp02] Configure the Offset Master TB on [ECU2].	
Step 8	[TSApp01] Get current time of the Offset Slave TB on [ECU1].	
Step 9	[TSApp01] Decide a future synchronization time based on the current time so that [TSApp01] and [TSApp02] will be notified simultaneously and sync then.	
Step 10	[TSApp01] Start the timer of the Offset Slave TB on [ECU1] so that the timer will expire at the synchronization time.	
Step 11	[TSApp01] Send the synchronization time to [TSApp02].	
Step 12	[TSApp02] Receive the synchronization time from [TSApp01].	
Step 13	[TSApp02] Get current time of the Offset Master TB on [ECU2].	
Step 14	[TSApp02] Start the timer of the Offset Master TB on [ECU2] so that the timer will expire at the synchronization time.	
Step 15	[TSApp01][TSApp02] Receive notify from the timer at the synchronization time.	
Step 16	[TSApp01][TSApp02] Get the current time and store the current time.	Both current times are almost same.

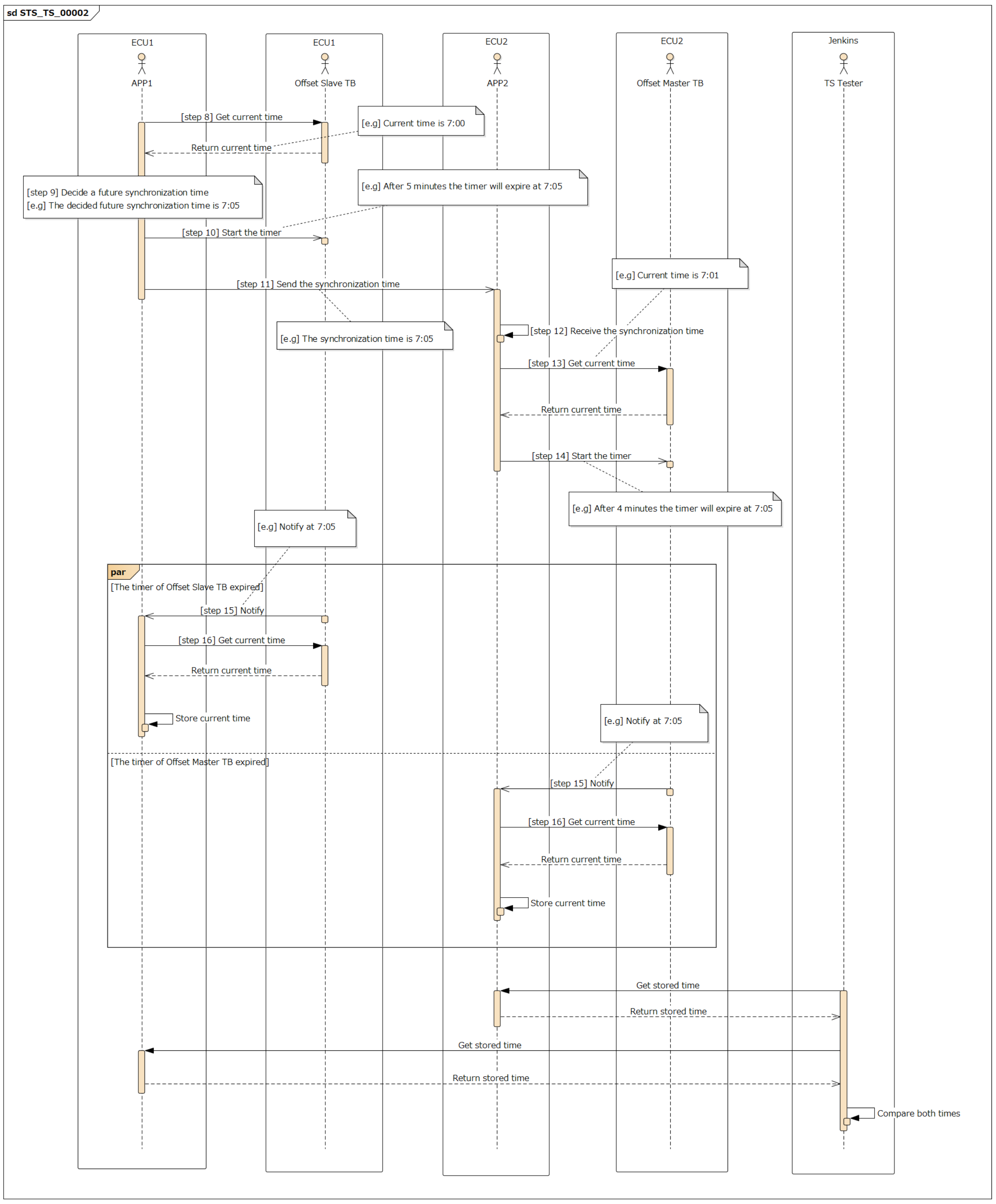
sd STS_TS_00002
ECU1 APP1
ECU1 Offset Slave TB
ECU2 APP2
ECU2 Offset Master TB
Jenkins TS Tester
[step 8] Get current time
Return current time
[e.g] Current time is 7:00
[step 9] Decide a future synchronization time
[e.g] The decided future synchronization time is 7:05
[e.g] After 5 minutes the timer will expire at 7:05
[step 10] Start the timer
[step 11] Send the synchronization time
[e.g] Current time is 7:01
[e.g] The synchronization time is 7:05
[step 12] Receive the synchronization time
[step 13] Get current time
Return current time
[step 14] Start the timer
[e.g] After 4 minutes the timer will expire at 7:05
[e.g] Notify at 7:05
par
[The timer of Offset Slave TB expired]
[step 15] Notify
[step 16] Get current time
Return current time
Store current time
[e.g] Notify at 7:05
[The timer of Offset Master TB expired]
[step 15] Notify
[step 16] Get current time
Return current time
Store current time
Get stored time
Return stored time
Get stored time
Return stored time
Compare both times

Figure 12.2: Sequence diagram of STS_TS_00002. [e.g] TSApp01 and TSApp02 sync at 7:05.

12.2.3 [STS_TS_00003] Check APIs of Offset Master TimeBase (TB) which do not impact other TB.

Test Objective	Verification that whether APIs of Offset Master TB can be used correctly.		
ID	STS_TS_00003	**State**	Draft
Affected Functional Cluster	Time Synchronization		
Trace to RS Criteria	[RS_TS_00005], [RS_TS_00012], [RS_TS_00013], [RS_TS_00017], [RS_TS_00026], [RS_TS_00029]		
Reference to Test Environment	STC_TS_00001 in Test configurations		
Configuration Parameters	- [ECU2] is Global Time Master. - [ECU2] has a Offset Master TB and a Synchronized Master TB. - The Offset Master TB on [ECU2] depend on the Synchronized Master TB on [ECU2].		
Summary	Test case 3 calls APIs of Offset Master TB on [ECU2] and confirms whether it works properly. The test scope is APIs which impact only Offset Master TB on [ECU2], do not impact Sync Master TB on [ECU2].		
Pre-conditions	- [TS Tester] is connected to [ECU2]. - [ECU2] is in Machine State Parking. - [TSApp02] is shut down according to Machine State.		
Post-conditions	[TS Tester] is disconnected to [ECU2].		
Main Test Execution			
Test Steps		**Pass Criteria**	
Step 1	[TS Tester] Request for change of Machine State to Driving from Execution Manager. Machine State for [ECU2] is changed to Driving, and [TSApp02] is started up.		
Step 2	[TSApp02] Find the Offset Master TB on [ECU2].	The Offset Master TB on [ECU2] is found successfully.	
Step 3	[TSApp02] Find the Synch Master TB on [ECU2].	The Synch Master TB on [ECU2] is found successfully.	
Step 4	[TSApp02] Get a getType of the Offset Master TB on [ECU2].	The getType is Offset Master TB.	
Step 5	[TSApp02] Set Offset value of the Offset Master TB on [ECU2].		
Step 6	[TSApp02] Get Offset value of the Offset Master TB on [ECU2].	Offset value is the value set in Step 5.	
Step 7	[TSApp02] Get current time of the Synch Master TB on [ECU2].	Current time is got successfully.	
Step 8	[TSApp02] Get current time of the Offset Master TB on [ECU2].	Current time is approximately that Offset value got in Step 6 added time value got in Step 7.	

▽

△

Step 9	[TSApp02] Start the timer of the Offset Master TB on [ECU2], so that the timer will expire at the specified time.	
Step 10	[TSApp02] When time-up is notified. Get current time of the Offset Master TB on [ECU2].	Current time is the specified time.

12.2.4 [STS_TS_00004] Check APIs of Offset Master TB which impact Sync Master TB.

Test Objective	Verification that APIs of Offset Master TB which impact Sync Master TB work properly and APIs of Time Base Status of Offset Master TB work properly.		
ID	STS_TS_00004	**State**	Draft
Affected Functional Cluster	Time Synchronization		
Trace to RS Criteria	[RS_TS_00010], [RS_TS_00014], [RS_TS_00015], [RS_TS_00018], [RS_TS_00021], [RS_TS_00026]		
Reference to Test Environment	STC_TS_00001 in Test configurations		
Configuration Parameters	- [ECU2] is Global Time Master. - [ECU2] has a Offset Master TB and a Synchronized Master TB. - The Offset Master TB on [ECU2] depend on the Synchronized Master TB on [ECU2].		
Summary	Set rate correction of Offset Master TB and confirm it is reflected by the value of rate deviation of Offset Master TB and Sync Master TB . Set Global time of Offset Master TB and confirm it is reflected by Offset Master TB and Sync Master TB. Set User data of Offset Master TB and confirm it is reflected by Offset Master TB and Sync Master TB. Get Time Base Status by calling API and confirm that It is got successfully.		
Pre-conditions	- [TS Tester] is connected to [ECU2]. - [ECU2] is in Machine State Parking. - [TSApp02] is shut down according to Machine State.		
Post-conditions	[TS Tester] is disconnected to [ECU2].		
Main Test Execution			
Test Steps		**Pass Criteria**	
Step 1	[TS Tester] Request for change of Machine State to Driving from Execution Manager. Machine State for [ECU2] is changed to Driving, and [TSApp02] is started up.		
Step 2	[TSApp02] Find the Offset Master TB on [ECU2].	The Offset Master TB on [ECU2] is found successfully.	
Step 3	[TSApp02] Find the Synch Master TB on [ECU2].	The Synch Master TB on [ECU2] is found successfully.	

▽

△

Step 4	[TSApp02] Set rate correction of the Offset Master TB on [ECU2].	
Step 5	[TSApp02] Get rate deviation of the Offset Master TB on [ECU2].	The value of rate deviation is the value set in Step 4 minus one.
Step 6	[TSApp02] Get rate deviation of the Synch Master TB on [ECU2].	The value of rate deviation is the value set in Step 4 minus one.
Step 7	[TSApp02] Set Global time of the Offset Master TB on [ECU2] by API of <SetTime>.	
Step 8	[TSApp02] Get current time of the Offset Master TB on [ECU2].	The time is approximately the value set in step 7.
Step 9	[TSApp02] Get current time of the Synch Master TB on [ECU2].	The time is approximately the value set in step 7.
Step 10	[TSApp02] Set Global time of the Offset Master TB on [ECU2] by API of <UpdateTime>.	
Step 11	[TSApp02] Get current time of the Offset Master TB on [ECU2].	The time is approximately the value set in step 10.
Step 12	[TSApp02] Get current time of the Synch Master TB on [ECU2].	The time is approximately the value set in step 10.
Step 13	[TSApp02] Set User Data of the Offset Master TB on [ECU2].	
Step 14	[TSApp02] Get Time Base Status of the Offset Master on [ECU2].	Time Base Status is got successfully.
Step 15	[TSApp02] Get User Data of the Time Base Status of the Offset Master on [ECU2].	The value of User Data is the value set in Step 13.
Step 16	[TSApp02] Get Update Counter of the Time Base Status of the Offset Master on [ECU2].	Update Counter is got successfully.
Step 17	[TSApp02] Get Synch Status of the Time Base Status of the Offset Master on [ECU2].	Synch Status is got successfully.
Step 18	[TSApp02] Get Status Flag of the Time Base Status of the Offset Master on [ECU2].	Status Flag is got successfully.
Step 19	[TSApp02] Get Creation Time of the Time Base Status of the Offset Master on [ECU2].	Creation Time is got successfully.

▽

△

Step 20	[TSApp02] Get Time Leap of the Time Base Status of the Offset Master on [ECU2].	Time Leap is got successfully.
Step 21	[TSApp02] Get Time Base Status of the Sync Master on [ECU2].	Time Base Status is got successfully.
Step 22	[TSApp02] Get User Data of the Time Base Status of the Sync Master on [ECU2].	The value of User Data is the value set in Step 13. User data is common value between Offset Master TB and Sync Master TB.

12.2.5 [STS_TS_00005] Check APIs of Offset Master TB which impact Offset Slave TB on the other ECU.

Test Objective	Verification that APIs of setting Global Time and User data work properly.		
ID	STS_TS_00005	**State**	Draft
Affected Functional Cluster	Time Synchronization		
Trace to RS Criteria	[RS_TS_00007], [RS_TS_00010], [RS_TS_00011], [RS_TS_00015], [RS_TS_00021], [RS_TS_00026]		
Reference to Test Environment	STC_TS_00001 in Test configurations		
Configuration Parameters	- [ECU1] is synced by [ECU2]. - [ECU2] is Global Time Master. - [ECU1] has a Offset Slave TimeBase(TB) and a Synchronized Slave TB. - [ECU2] has a Offset Master TB and a Synchronized Master TB. - The Synchronized Slave TB on [ECU1] is synced by the Synchronized Master TB on [ECU2]. - The Offset Slave TB on [ECU1] depend on the Synchronized Slave TB on [ECU1], - The Offset Master TB on [ECU2] depend on the Synchronized Master TB on [ECU2]. - Event based communication. - The existing communication services comprise the following (service & data names are arbitrary): ● [TSService01]: Offered by [TSApp02], requested by [TSApp01]. ● [TSService01]: [TSApp02] send a global time and user data to [TSApp01].		
Summary	Set User data of Offset Master TB and confirm it is reflected by Offset Master TB on [ECU2] and Offset Slave TB on [ECU1]. User data is sent from Master TB to Slave TB. Set Global time of Offset Master TB and confirm it is reflected by Offset Master TB on [ECU2] and Offset Slave TB on [ECU1].		
Pre-conditions	- [TS Tester] is connected to both ECUs. - Both ECUs are in Machine State Parking. - [TSApp01] and [TSApp02] are shut down according to Machine State.		
Post-conditions	[TS Tester] is disconnected to both ECUs.		
Main Test Execution			
Test Steps		**Pass Criteria**	

▽

△

Step 1	[TS Tester] Request for change of Machine State to Driving from Execution Manager. Machine State for [ECU1] and [ECU2] are changed to Driving, and [TSApp01] and [TSApp02] are started up.	
Step 2	[TSApp02] Offer service [TSService01].	
Step 3	[TSApp01] Request service [TSService01].	
Step 4	[TSApp02] Find the Offset Master TB on [ECU2].	The Offset Master TB on [ECU2] is found successfully.
Step 5	[TSApp01] Find the Offset Slave TB on [ECU1].	The Offset Slave TB on [ECU1] is found successfully.
Step 6	[TSApp02] Set User Data of the Offset Master TB on [ECU2].	
Step 7	[TSApp02] Get Time Base Status of the Offset Master TB on [ECU2].	Time Base Status is got successfully.
Step 8	[TSApp02] Get User Data of Time Base Status of the Offset Master TB on [ECU2].	The value of User Data is the value set in Step 6.
Step 9	[TSApp02] Set a Global time of the Offset Master TB by API of <SetTime>.	
Step 10	[TSApp02] Get current time of the Offset Master TB on [ECU2].	Current time is approximately the value set in step 9.
Step 11	[TSApp02] The Global time set in step 9 and User data set in step 6 is sent to [TSApp01] and wait until [TSApp01] has confirmed Global time and User Data.	
Step 12	[TSApp01] Receive a set Global time and User Data from [TSApp02].	
Step 13	[TSApp01] Get Time Base Status of the Offset Slave TB on [ECU1].	Time Base Status is got successfully.
Step 14	[TSApp01] Get User Data of Time Base Status of the Offset Slave TB on [ECU1].	The value of User Data is the value set in Step 6. User data is common value between Master TB on [ECU2] and Slave TB on [ECU1].
Step 15	[TSApp01] Get current time of the Offset Slave TB on [ECU1].	Current time is approximately the value set in step 9.
Step 16	[TSApp02] Set a Global time of the Offset Master TB by API of <UpdateTime>.	

▽

△

Step 17	[TSApp02] Get current time of the Offset Master TB on [ECU2].	Current time is approximately the value set in step 16.
Step 18	[TSApp02] The set Global time is sent to [TSApp01].	Both current times are almost same.
Step 19	[TSApp01] Receive a set global time from [TSApp02] and wait until Global Time on [ECU1] has been updated.	
Step 20	[TSApp01] Get current time of the Offset Slave TB on [ECU1].	Current time is approximately the value set in step 16.

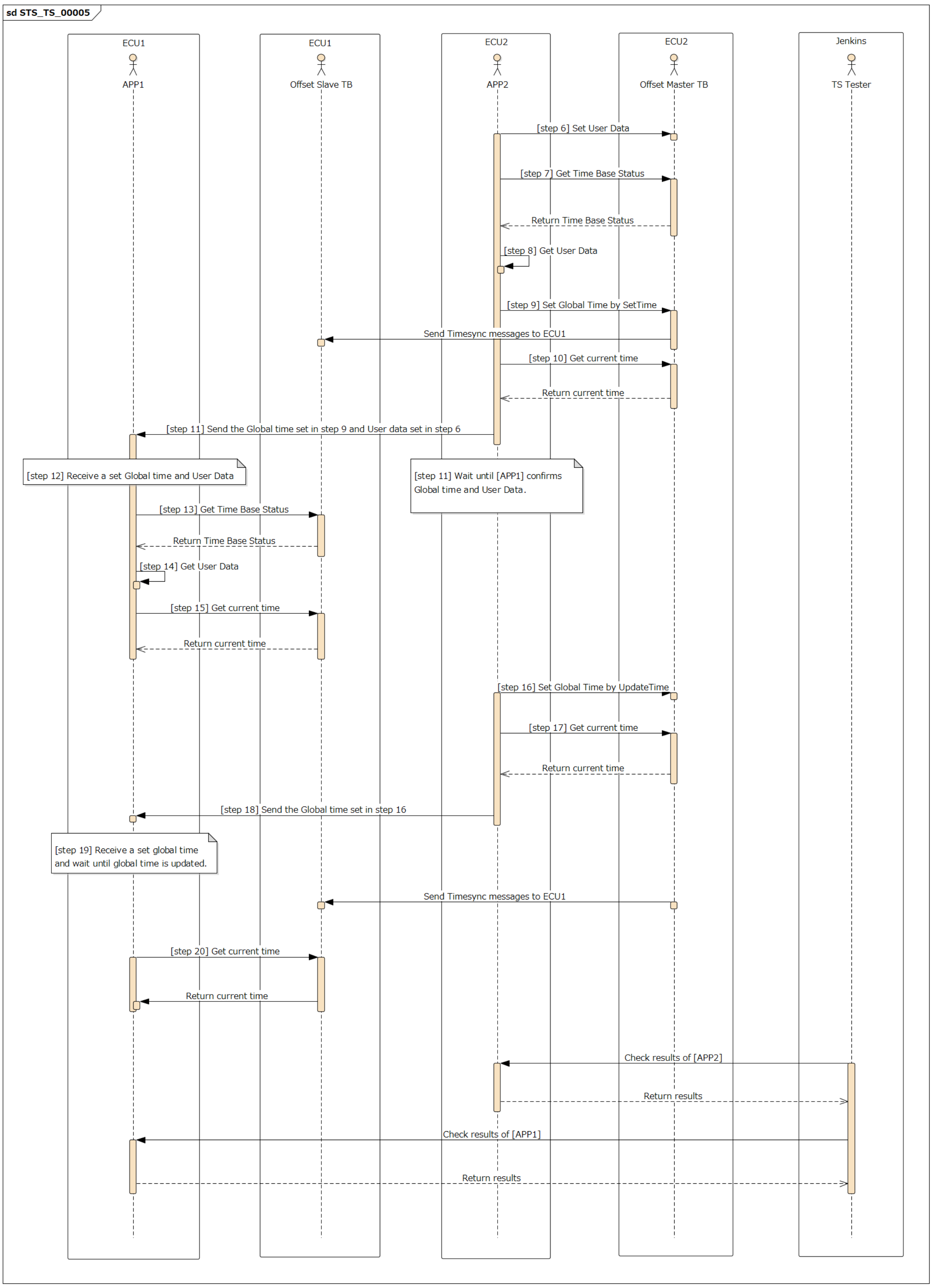
sd STS_TS_00005
ECU1
APP1
ECU1
Offset Slave TB
ECU2
APP2
ECU2
Offset Master TB
Jenkins
TS Tester
[step 6] Set User Data
[step 7] Get Time Base Status
Return Time Base Status
[step 8] Get User Data
[step 9] Set Global Time by SetTime
Send Timesync messages to ECU1
[step 10] Get current time
Return current time
[step 11] Send the Global time set in step 9 and User data set in step 6
[step 12] Receive a set Global time and User Data
[step 11] Wait until [APP1] confirms Global time and User Data.
[step 13] Get Time Base Status
Return Time Base Status
[step 14] Get User Data
[step 15] Get current time
Return current time
[step 16] Set Global Time by UpdateTime
[step 17] Get current time
Return current time
[step 18] Send the Global time set in step 16
[step 19] Receive a set global time and wait until global time is updated.
Send Timesync messages to ECU1
[step 20] Get current time
Return current time
Check results of [APP2]
Return results
Check results of [APP1]
Return results

Figure 12.3: Sequence diagram of STS_TS_00005.

13 Test configuration and test steps for Security Management

13.1 Test System

Security Management is responsible for aspects related to Secure Communication and Protected Runtime Environment.

The purpose of Secure Communication is to ensure message confidentiality, integrity and authentication. These capabilities are offered as a library to facilitate reusability.

Protected Runtime Environment ensures inter-process separation (spatial, time and resource) and protection against memory corruption attacks.

System Tests target to check successful communication of messages using secure channels, irrespective of underlying libraries and cypher suites.

13.1.1 Test configurations

Configuration ID	STC_SEC_00001
Description	Standard Jenkins server for Security test
ECU 1	Hardware, 192.168.100.5
ECU 2	Hardware, 192.168.100.2
Jenkins	Jenkins Server, 192.168.100.10

Jenkins Server, running the job with Security Tester is connected via Ethernet to [ECU1] hosting the Security Test Application (STA) and [ECU2].

[ECU1] sends the data to [ECU2]. Man-in-middle attack is performed through Jenkins Server.

The Security Tester is supposed to check pass criteria.

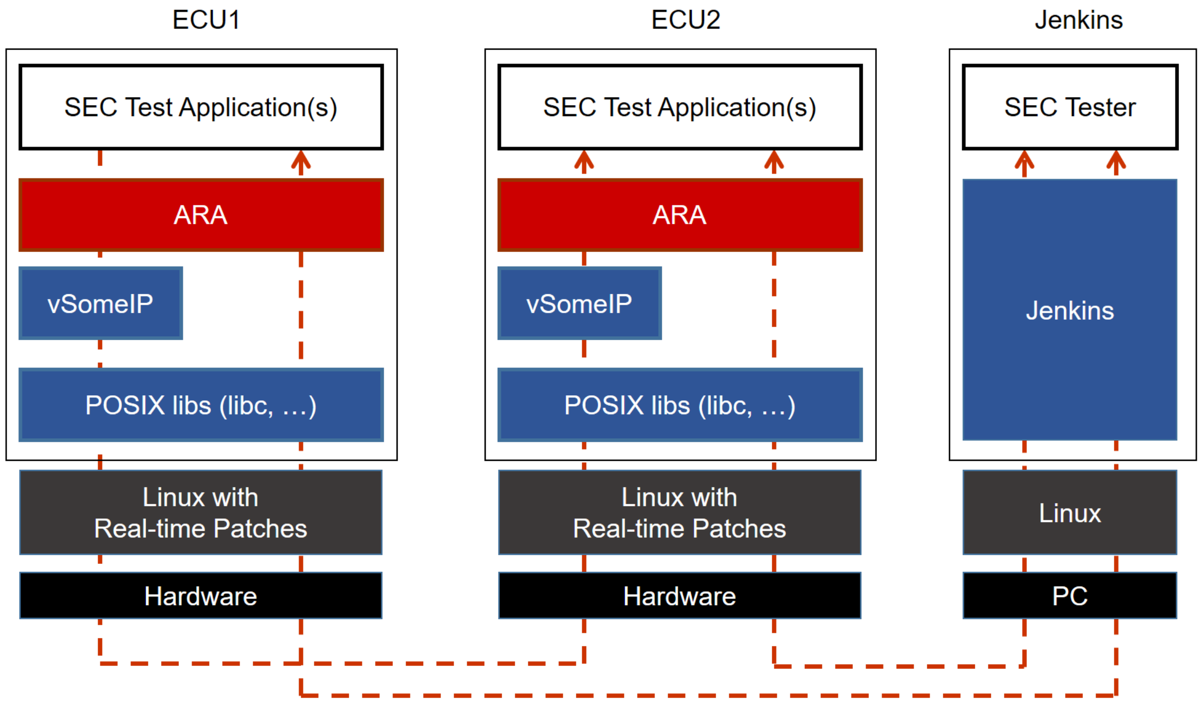

Figure 13.1: Illustration of test setup for Security Management.

13.2 Test cases for Secure Communication

13.2.1 [STS_SEC_00001] Message authentication

Test Objective	Verification that the messages from only authentic source are considered and replay attacks are prevented.		
ID	STS_SEC_00001	**State**	Draft
Affected Functional Cluster	Security		
Trace to RS Criteria	[RS_SEC_04001], [RS_SEC_04002], [RS_SEC_04003], [RS_SEC_04004]		
Reference to Test Environment	STC_SEC_00001 in Test configurations		
Configuration Parameters	- Secure channels and cypher suites are peoperly configured in the manifest. - Secure channel configurations for the applications are provided by manifests.		
Summary	This test case aims to verify that - Messages are securely transferred from sender [ECU1] to the receiver [ECU2] - Messages are successfully authenticated and verified - Any replay attacks are unsuccessful		

▽

△

Pre-conditions	- Security Tester is connected to [ECU1] and [ECU2] - Software components on [ECU1] and [ECU2] are initialized - Secure channel between [SECApp01] on [ECU1] and [SECApp02] on [ECU2] exists - [ATTACKER] is configured on Jenkins to listen to the same port as [SECApp02]		
Post-conditions	TCP connections between Security Tester and [ECU1] and [ECU2] is closed.		
Main Test Execution			
Test Steps		**Pass Criteria**	
Step 1	[SECApp01] Create a payload "Hello World" and send using secure channel to [SECApp02]		
Step 2	[SECApp02] Receive message and try to authenticate	Message authentication successful, which means message received from [SECApp01]	
Step 3	[ATTACKER] Perform replay attack by sending message "Hello World" to [SECApp02]		
Step 4	[SECApp02] Receive message and try to authenticate	Message authentication fails which means message was not sent by [SECApp01]. Message is discarded and replay attack is unsuccessful.	

13.2.2 [STS_SEC_00002] Message confidentiality and integrity

Test Objective	Verification that only authorized source can decrypt a message and the message integrity is maintained.		
ID	STS_SEC_00002	**State**	Draft
Affected Functional Cluster	Security		
Trace to RS Criteria	[RS_SEC_04001], [RS_SEC_04002], [RS_SEC_04003], [RS_SEC_04004]		
Reference to Test Environment	STC_SEC_00001 in Test configurations		
Configuration Parameters	- Secure channels and cypher suites are peoperly configured in the manifest. - Secure channel configurations for the applications are provided by manifests.		
Summary	This test case aims to verify that - Messages are securely transferred from sender [ECU1] to the receiver [ECU2] - Messages are successfully authenticated and verified - Decryption and tempering of message is unsuccessful		
Pre-conditions	- Security Tester is connected to [ECU1] and [ECU2] - Software components on [ECU1] and [ECU2] are initialized - Secure channel between [SECApp01] on [ECU1] and [SECApp02] on [ECU2] exists - [ATTACKER] is configured on Jenkins to listen to the same port as [SECApp02]		
Post-conditions	TCP connections between Security Tester and [ECU1] and [ECU2] is closed.		
Main Test Execution			
Test Steps		**Pass Criteria**	

▽

△

Step 1	[SECApp01] Create a payload "Hello World" and send plain text to [TESTER]	Message "Hello World" received by [TESTER]
Step 2	[SECApp01] Send the same payload using secure channel to [SECApp02]	Encrypted messaged received by [SECApp02]
Step 3	[SECApp02] Authenticate the messaged received from [SECApp01]	Message authentication successful, which means message received from [SECApp01]
Step 4	[SECApp02] Decrypt message from [SECApp01]	Message decrypted as "Hello World". Message integrity is proved.
Step 5	[SECApp02] Send decrypted message to [TESTER]	"Hello World" received by [TESTER] and is stored for further comparison
Step 6	[ATTACKER] Sniff the message sent over secure channel from [SECApp01] to [SECApp02]	Encrypted message received by [ATTACKER]
Step 7	[ATTACKER] Try to decrypt message sniffed earlier	Decryption attempt unsuccessful. Message confidentiality is proven.
Step 8	[ATTACKER] If the decryption was successful (by guessing the key or if encryption was weak), then send decrypted message to [TESTER], else send sniffed (encrypted) message to [TESTER]	Message received by [TESTER] and is stored for further comparison
Step 9	[TESTER] Compare plain text from [SECApp01] and decrypted message from [SECApp02]	Both messages are exactly same. Message integrity is proved.
Step 10	[TESTER] Compare plain text from [SECApp01] and encrypted/ decrypted message from [ATTACKER]	Both messages are different. Message confidentiality is proved.

14 Test configuration and test steps for Network Management

14.1 Test System

14.1.1 Test configurations NM

Configuration ID	STC_NM_00001
Description	Scenario 1 - All ECUs are in the same NM Cluster
ECU 1	Hardware, 192.168.100.5
ECU 2	Hardware, 192.168.100.2
ECU 3	Hardware, 192.168.100.3
Jenkins	Jenkins Server, 192.168.100.10

Configuration ID	STC_NM_00002
Description	Scenario 2 - only ECU2 is in the NM cluster
ECU 1	Hardware, 192.168.100.5
ECU 2	Hardware, 192.168.100.2
ECU 3	Hardware, 192.168.100.3
Jenkins	Jenkins Server, 192.168.100.10

The Jenkins Server, running the job with the Network Management test [NM TESTER] is connected via Ethernet to [ECU1] hosting the NM Test Application [NMApp01], [ECU2] hosting the NM Test Application [NMApp02] and [ECU3] hosting the NM Test Application [NMApp03].

The [NM Tester] is supposed to collect the results by checking multicast messages.

The communication between [NM Tester] and the applications on the ECU may take place over the Diagnostics functional cluster in form of diagnostic messages.

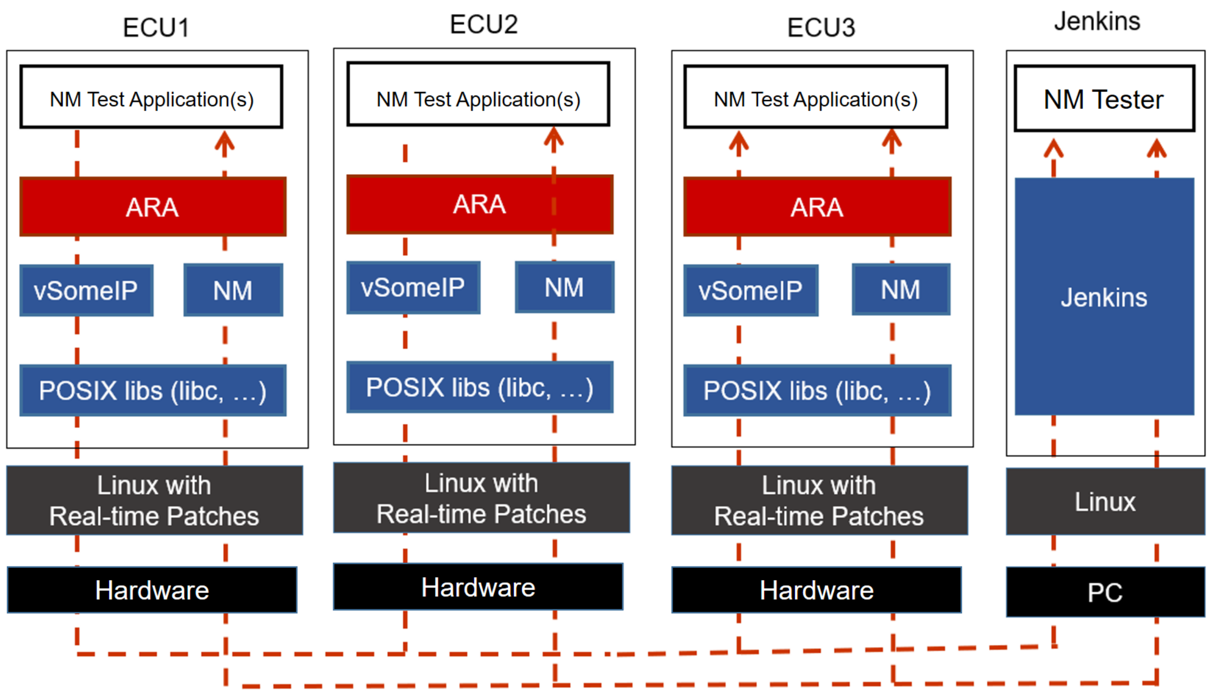

Figure 14.1: Illustration of test setup for Network Management

14.2 Test cases Network Management

14.2.1 [STS_NM_00001] Basic Network Management functionality of ECUs in same NM Cluster.

Test Objective	To verify that the Basic Network Management functionality of ECUs in same NM Cluster works.		
ID	STS_NM_00001	**State**	Draft
Affected Functional Cluster	NM		
Trace to RS Criteria	[RS_Nm_00044], [RS_Nm_00047], [RS_Nm_00048], [RS_Nm_00050], [RS_Nm_00054]		
Reference to Test Environment	STC_NM_00001 in Test configurations NM		
Configuration Parameters	NM configuration parameters are configured		
Summary	Initially all three ECUs are in inactive state. Machine state of [ECU2] is changed to Driving. [ECU2] sends multicast NM messages periodically which is received by [ECU1] and [ECU3] and due to this [ECU1] and [ECU3] become active. Network change its mode from Bus sleep mode to Network Mode. [ECU2] stops sending NM messages and becomes inactive. [ECU1] and [ECU3] does not receive NM messages for a time <t> and [ECU1] becomes inactive. Network transitions its modes as per configured timeouts.		

▽

△

Pre-conditions	- [NM Tester] is connected to all ECUs. - All ECUs are in Machine State Parking. - Applications are shut down according to Machine State.	
Post-conditions	TCP connections between [NM Tester] and all ECUs are closed.	
Main Test Execution		
Test Steps		**Pass Criteria**
Step 1	[NM TESTER] Check Network Current State.	Field Network-State.NetworkCurrentState is set to false.
Step 2	[NM TESTER] Request the change of Machine State to Driving for ECU2.	Machine State for ECU2 is changed to Driving.
Step 3	[NMApp02] Request NM to send multicast messages.	
Step 4	[NM TESTER] Check NM multicast messages	Multicast messages are received with source node ID of [ECU2] with logical network information bit set to 1. [ECU1] and [ECU3] become awake. Network enters into Network Mode (Repeat Message State).
Step 5	[NM TESTER] Check NM multicast messages after <Repeat Message timer> expired	Network enters into Network Mode (Normal Operation State).
Step 6	[NM TESTER] Check Network Current State.	Field Network-State.NetworkCurrentState is set to true.
Step 7	[NM TESTER] Check NM multicast messages after <NM-timeout timer> if all ECUs are still awake	Multicast messages are received with source node ID of [ECU2] [ECU1] and [ECU3] are awake.
Step 8	[NMApp02] Indicate NM to release the network to stop sending multicast message.	
Step 9	[NM TESTER] Check NM multicast messages.	Multicast messages are not received with source node ID of [ECU2] and Network goes to Ready Sleep state
Step 10	[NM TESTER] Check NM multicast messages after NM Timout timer <t>	Network goes to Prepare Bus sleep Mode.
Step 11	[NM TESTER] Check NM multicast messages after wait bus sleep timer <t>	Network goes to Bus sleep Mode.
Step 12	[NM TESTER] Check Network Current State.	Field Network-State.NetworkCurrentState is set to false.

14.2.2 [STS_NM_00002] Basic Network Management functionality of ECUs not in same partial network Cluster.

Test Objective	To verify that the Basic Network Management functionality of ECUs not in same partial network Cluster works.		
ID	STS_NM_00002	**State**	Draft
Affected Functional Cluster	NM		
Trace to RS Criteria	[RS_Nm_00044], [RS_Nm_00047], [RS_Nm_00048], [RS_Nm_02517], [RS_Nm_00050], [RS_Nm_00054]		
Reference to Test Environment	STC_NM_00002 in Test configurations NM		
Configuration Parameters	NM configuration parameters are configured		
Summary	Initially all three ECUs are in inactive state. [ECU1] and [ECU2] forms a partial network. Machine state of [ECU2] is changed to Driving. [ECU2] sends multicast NM messages periodically which is received by [ECU1] but [ECU3] ignores it and due to this [ECU1] becomes active while [ECU3] remains inactive. Network change its mode from Bus sleep mode to Network Mode. [ECU2] stops sending NM messages and becomes inactive. [ECU1] and [ECU3] does not receive NM messages for a time <t1> and [ECU1] becomes inactive. Network transitions its modes as per configured timeouts.		
Pre-conditions	- [NM Tester] is connected to all the ECUs. - All ECUs are in Machine State Living. - Applications are shut down according to Machine State.		
Post-conditions	TCP connections between [NM Tester] and both ECUs are closed.		
Main Test Execution			
Test Steps			**Pass Criteria**
Step 1	[NM TESTER] Check Network Current State for the Partial Network.		Field Network-State.NetworkCurrentState is set to false.
Step 2	[NM TESTER] Request the change of Machine State to Driving for ECU2.		Machine State for ECU2 is changed to Driving.
Step 3	[NMApp02] Request NM to send multicast messages.		
Step 4	[NM TESTER] Check NM multicast messages		Multicast messages are received with source node ID of [ECU2] with logical network information bit set to 1. [ECU1] becomes awake and [ECU3] ignores it and remains inactive. Network enters into Network Mode (Repeat Message State).
Step 5	[NM TESTER] Check NM multicast messages after <Repeat Message timer> expired		Network enters into Network Mode (Normal Operation State).

▽

△

Step 6	[NM TESTER] Check NM multicast messages after <NM-timeout timer> if [ECU2] is awake and [ECU3] is in sleep.	Multicast messages are received with source node ID of [ECU2] [ECU1] is awake while [ECU3] remains inactive. NM message is received from [ECU1]
Step 7	[NM TESTER] Check Network Current State of Partial Network.	Field Network-State.NetworkCurrentState is set to true.
Step 8	[NMApp02] Indicate NM to release the network to stop sending multicast message.	
Step 9	[NM TESTER] Check NM multicast messages.	Multicast messages are not received with source node ID of [ECU2] and Network goes to Ready Sleep state
Step 10	[NM TESTER] Check NM multicast messages after NM Timout timer <t1>	Network goes to Prepare Bus sleep Mode.
Step 11	[NM TESTER] Check NM multicast messages after wait bus sleep timer <t2>	Network goes to Bus sleep Mode.
Step 12	[NM TESTER] Check Network Current State of Partial Network.	Field Network-State.NetworkCurrentState is set to false.

15 Test configuration and test steps for Cryptography

15.1 Test System

15.1.1 Test configurations

Configuration ID	STC_CRYPTO_00001
Description	Standard Jenkins server for Cryptography test
ECU 1	Hardware, 192.168.100.2
Jenkins	Jenkins Server, 192.168.100.10

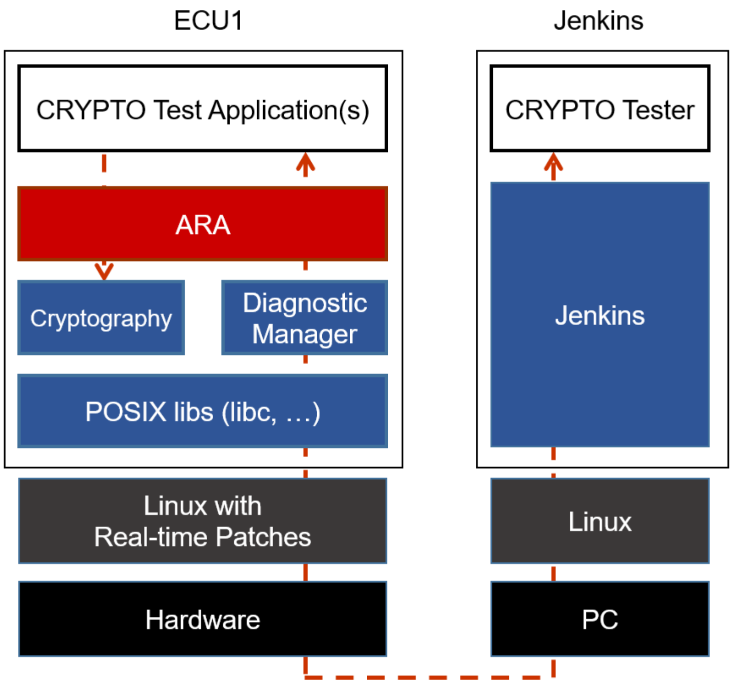

Figure 15.1: Illustration of test setup for Cryptography.

The Jenkins Server, running the job with the Cryptography test ([CRYPTO Tester]) is connected via Ethernet to [ECU1] hosting the CRYPTO Test Applications [CRYPTOApp01].

The [CRYPTO Tester] is supposed to check the pass criteria.

The communication between [CRYPTO Tester] and the [CRYPTOApp01] may take place over the Diagnostics functional cluster in form of diagnostic messages.

15.2 Test cases

15.2.1 [STS_CRYPTO_00001] Encrypting and decrypting data using an algorithm for symmetric encryption/decryption primitives.

Test Objective	Verify that Crypto Stack correctly encrypts and decrypts data using symmetric key.		
ID	STS_CRYPTO_00001	**State**	Draft
Affected Functional Cluster	Cryptography		
Trace to RS Criteria	[RS_CRYPTO_02001], [RS_CRYPTO_02008], [RS_CRYPTO_02201], [RS_CRYPTO_02302]		
Reference to Test Environment	STC_CRYPTO_00001 in Test configurations		
Configuration Parameters	- Provide key for symmetric encryption/decryption. - Allow use of symmetric key for encryption and decryption by [CRYPTOApp01].		
Summary	[CRYPTO Tester] sends <Plaintext1> to [CRYPTOApp01] and is encrypted on the [CRYPTOApp01] side using symmetric key <SK1> to obtain <Ciphertext1'>. <Ciphertext1'> is compared with <Ciphertext1> which is generated in the same way on the [CRYPTO Tester] side. [CRYPTO Tester] sends <Ciphertext2> to [CRYPTOApp01] and is decrypted on the [CRYPTOApp01] side to obtain <Plaintext2'>. <Plaintext2'> is compared with <Plaintext2> on the [CRYPTO Tester] side. - Data encryption/decryption on the [CRYPTO Tester] side is performed either prior to running test or during a test step. - Whether to compare encryption/decryption result (<Ciphertext1'> and <Plaintext2'>) in [CRYPTOApp01] or [CRYPTO Tester] is up to implementer.		
Pre-conditions	- Crypto stack and [CRYPTOApp01] are initialized with used key (<SK1>), algorithm, and domain parameter as applicable. - Communication between [CRYPTO Tester] and [CRYPTOApp01] has been set up. - Symmetric key <SK1> can be accessed by [CRYPTOApp01].		
Post-conditions	Communication between [CRYPTO Tester] and [CRYPTOApp01] is closed.		
Main Test Execution			
Test Steps		**Pass Criteria**	
Step 1	[CRYPTO Tester] Send <Plaintext1> to [CRYPTOApp01].		
Step 2	[CRYPTOApp01] Encrypt <Plaintext1> using symmetric key <SK1> to obtain <Ciphertext1'>.		
Step 3	[CRYPTOApp01] Return <Plaintext1> encryption status to [CRYPTO Tester].		
Step 4	[CRYPTO Tester] Check encryption status.	[CRYPTO Tester] Encryption status contains success and no error.	
Step 5	[CRYPTO Tester] Send <Ciphertext1> (i.e. <Plaintext1> encrypted in the same way on the [CRYPTO Tester] side) to [CRYPTOApp01].		
Step 6	[CRYPTOApp01] Compare <Ciphertext1'> with <Ciphertext1>.		

▽

△

Step 7	[CRYPTOApp01] Return comparison result (matched/unmatched) to [CRYPTO Tester].	
Step 8	[CRYPTO Tester] Check comparison result.	[CRYPTO Tester] Comparison result is "matched."
Step 9	[CRYPTO Tester] Send <Ciphertext2> to [CRYPTOApp01].	
Step 10	[CRYPTOApp01] Decrypt <Ciphertext2> using symmetric key <SK1> to obtain <Plaintext2'>.	
Step 11	[CRYPTOApp01] Return <Ciphertext2> dencryption status to [CRYPTO Tester].	
Step 12	[CRYPTO Tester] Check decryption status.	[CRYPTO Tester] Decryption status contains success and no error.
Step 13	[CRYPTO Tester] Send <Plaintext2> to [CRYPTOApp01].	
Step 14	[CRYPTOApp01] Compare <Plaintext2'> with <Plaintext2>.	
Step 15	[CRYPTOApp01] Return comparison result (matched/unmatched) to [CRYPTO Tester].	
Step 16	[CRYPTO Tester] Check comparison result.	[CRYPTO Tester] Comparison result is "matched."

15.2.2 [STS_CRYPTO_00002] Encrypting and decrypting data using an algorithm for asymmetric encryption/decryption primitives.

Test Objective	Verify that Crypto Stack correctly encrypts and decrypts data using public and private keys.		
ID	STS_CRYPTO_00002	**State**	Draft
Affected Functional Cluster	Cryptography		
Trace to RS Criteria	[RS_CRYPTO_02002], [RS_CRYPTO_02008], [RS_CRYPTO_02202], [RS_CRYPTO_02302]		
Reference to Test Environment	STC_CRYPTO_00001 in Test configurations		
Configuration Parameters	- Provide public and private key pair for tested asymmetric encryption/decryption algorithm. - Allow use of public and private key pair for encryption and decryption by [CRYPTOApp01].		

▽

△

Summary	[CRYPTO Tester] sends <Plaintext1> (up to maximum possible bit length for used algorithm) to [CRYPTOApp01] and is encrypted on the [CRYPTOApp01] side using [CRYPTOApp01]'s public key <APbK> to obtain <Ciphertext1'>. <Ciphertext1'> is compared with <Ciphertext1> which is generated in the same way on the [CRYPTO Tester] side. [CRYPTO Tester] sends <Ciphertext2> (encrypted using <APbK>) to [CRYPTOApp01] and is decrypted on the [CRYPTOApp01] side using [CRYPTOApp01]'s private key <APvK> to obtain <Plaintext2'>. <Plaintext2'> is compared with <Plaintext2> on the [CRYPTO Tester] side. - Data encryption/decryption on the [CRYPTO Tester] side is performed either prior to running test or during a test step. - Whether to compare encryption/decryption result (<Ciphertext1'> and <Plaintext2'>) in [CRYPTOApp01] or [CRYPTO Tester] is up to implementer.	
Pre-conditions	- Crypto stack and [CRYPTOApp01] are initialized with used key (<APbK>), algorithm, and domain parameter as applicable. - Communication between [CRYPTO Tester] and [CRYPTOApp01] has been set up. - Public and private key pair <APbK> and <APvK> can be accessed by [CRYPTOApp01].	
Post-conditions	Communication between [CRYPTO Tester] and [CRYPTOApp01] is closed.	
Main Test Execution		
Test Steps		**Pass Criteria**
Step 1	[CRYPTO Tester] Send <Plaintext1> to [CRYPTOApp01].	
Step 2	[CRYPTOApp01] Encrypt <Plaintext1> using [CRYPTOApp01]'s public key <APbK> to obtain <Ciphertext1'>.	
Step 3	[CRYPTOApp01] Return <Plaintext1> encryption status to [CRYPTO Tester].	
Step 4	[CRYPTO Tester] Check encryption status.	[CRYPTO Tester] Encryption status contains success and no error.
Step 5	[CRYPTO Tester] Send <Ciphertext1> (<Plaintext1> encrypted using <APbK> on the [CRYPTO Tester] side) to [CRYPTOApp01].	
Step 6	[CRYPTOApp01] Compare <Ciphertext1'> with <Ciphertext1>.	
Step 7	[CRYPTOApp01] Return comparison result (matched/unmatched) to [CRYPTO Tester].	
Step 8	[CRYPTO Tester] Check comparison result.	[CRYPTO Tester] Comparison result is "matched."
Step 9	[CRYPTO Tester] Send <Ciphertext2> (<Plaintext2> encrypted using <APbK> on the [CRYPTO Tester] side) to [CRYPTOApp01].	
Step 10	[CRYPTOApp01] Decrypt <Ciphertext2> using [CRYPTOApp01]'s private key <APvK> to obtain <Plaintext2'>.	

▽

△

Step 11	[CRYPTOApp01] Return <Ciphertext2> dencryption status to [CRYPTO Tester].	
Step 12	[CRYPTO Tester] Check decryption status.	[CRYPTO Tester] Decryption status contains success and no error.
Step 13	[CRYPTO Tester] Send <Plaintext2> to [CRYPTOApp01].	
Step 14	[CRYPTOApp01] Compare <Plaintext2'> with <Plaintext2>.	
Step 15	[CRYPTOApp01] Return comparison result (matched/unmatched) to [CRYPTO Tester].	
Step 16	[CRYPTO Tester] Check comparison result.	[CRYPTO Tester] Comparison result is "matched."

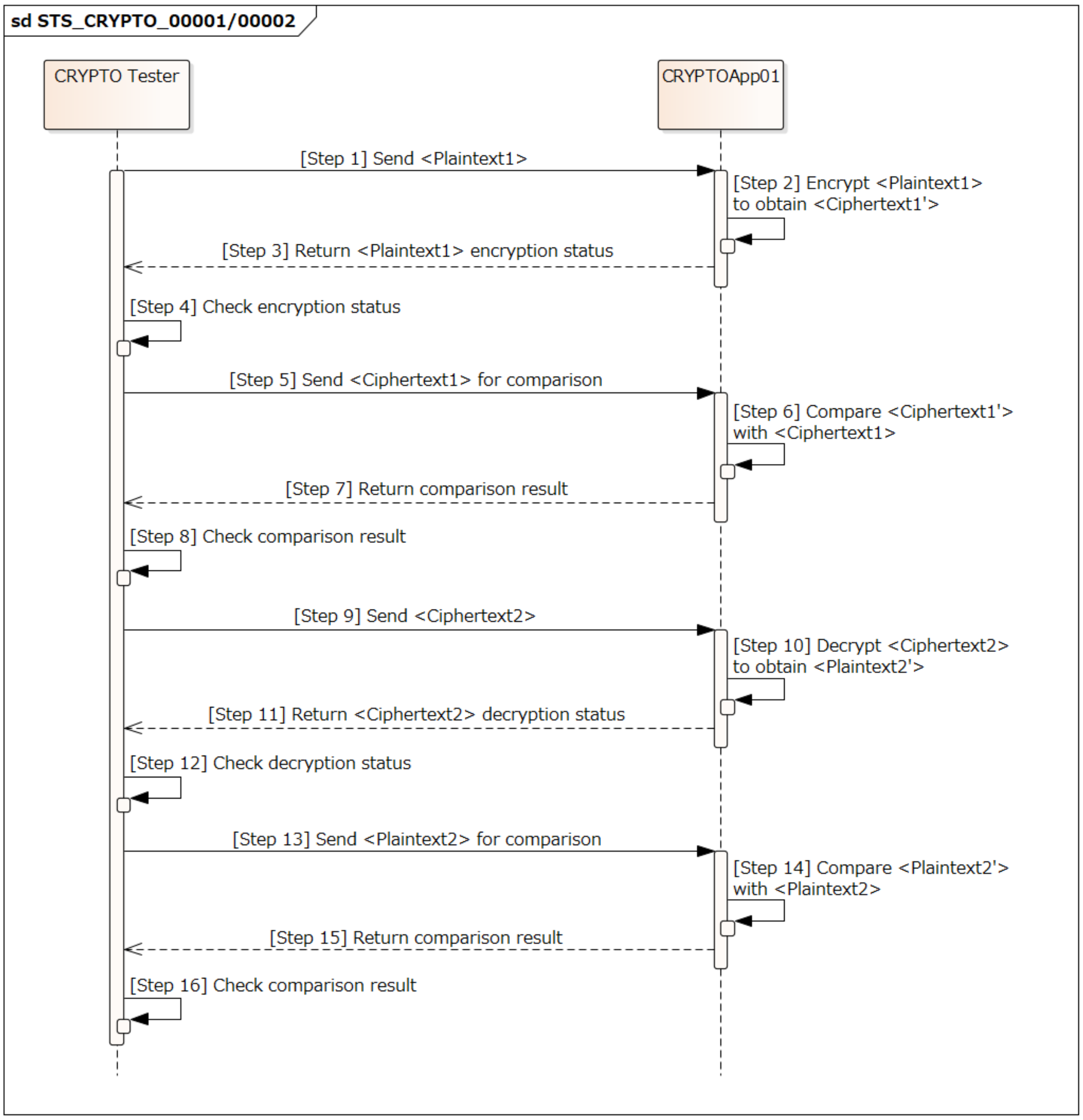
sd STS_CRYPTO_00001/00002
CRYPTO Tester
CRYPTOApp01
[Step 1] Send <Plaintext1>
[Step 2] Encrypt <Plaintext1> to obtain <Ciphertext1'>
[Step 3] Return <Plaintext1> encryption status
[Step 4] Check encryption status
[Step 5] Send <Ciphertext1> for comparison
[Step 6] Compare <Ciphertext1'> with <Ciphertext1>
[Step 7] Return comparison result
[Step 8] Check comparison result
[Step 9] Send <Ciphertext2>
[Step 10] Decrypt <Ciphertext2> to obtain <Plaintext2'>
[Step 11] Return <Ciphertext2> decryption status
[Step 12] Check decryption status
[Step 13] Send <Plaintext2> for comparison
[Step 14] Compare <Plaintext2'> with <Plaintext2>
[Step 15] Return comparison result
[Step 16] Check comparison result

Figure 15.2: Sequence diagram of STS_CRYPTO_00001/00002.

15.2.3 [STS_CRYPTO_00003] Generation and verification of message authentication code.

Test Objective	Verify that Crypto Stack correctly generates and verifies message authentication code.		
ID	STS_CRYPTO_00003	**State**	Draft
Affected Functional Cluster	Cryptograpny		

▽

△

Trace to RS Criteria	[RS_CRYPTO_02001], [RS_CRYPTO_02008], [RS_CRYPTO_02203], [RS_CRYPTO_02302]	
Reference to Test Environment	STC_CRYPTO_00001 in Test configurations	
Configuration Parameters	- Allow use of symmetric key <SK1> for generation of message authentication code by [CRYPTO Tester] and [CRYPTOApp01].	
Summary	[CRYPTO Tester] sends <Data1> to [CRYPTOApp01] and message authentication code <MAC1'> is generated by [CRYPTOApp01] from <Data1>. <MAC1'> is compared with <MAC1> which is generated in the same way on the [CRYPTO Tester] side. [CRYPTO Tester] sends <Data2> and <MAC2> (generated from <Data2> on the [CRYPTO Tester] side) to [CRYPTOApp01] and <MAC2> is compared by [CRYPTOApp01]. - Generation of <MAC1> and <MAC2> on the [CRYPTO Tester] side is performed either prior to running test or during a test step. - Whether to compare <MAC1'> in [CRYPTOApp01] or [CRYPTO Tester] is up to implementer.	
Pre-conditions	- Crypto stack and [CRYPTOApp01] are initialized with used key, algorithm, and domain parameter as applicable. - Communication between [CRYPTO Tester] and [CRYPTOApp01] has been set up.	
Post-conditions	Communication between [CRYPTO Tester] and [CRYPTOApp01] is closed.	
Main Test Execution		
Test Steps		**Pass Criteria**
Step 1	[CRYPTO Tester] Send <Data1> to [CRYPTOApp01].	
Step 2	[CRYPTOApp01] Generate message authentication code <MAC1'> from <Data1> (via MessageAuthnCodeCtx::Start()/Update()/Finish()).	
Step 3	[CRYPTOApp01] Return message authentication code generation status to [CRYPTO Tester].	
Step 4	[CRYPTO Tester] Check message authentication code generation status.	[CRYPTO Tester] Message authentication code generation status contains success and no error.
Step 5	[CRYPTO Tester] Send <MAC1> to [CRYPTOApp01].	
Step 6	[CRYPTOApp01] Compare <MAC1'> with <MAC1> (either by retrieving <MAC1'> with MessageAuthnCodeCtx::GetDigest() and compare with <MAC1>, or by passing <MAC1> to MessageAuthnCodeCtx::Compare()).	
Step 7	[CRYPTOApp01] Return comparison result (matched/unmatched) to [CRYPTO Tester].	
Step 8	[CRYPTO Tester] Check comparison result.	[CRYPTO Tester] Comparison result is "matched."

15.2.4 [STS_CRYPTO_00004] Generation and verification of digital signature.

Test Objective	Verify that Crypto Stack correctly generates and verifies digital signature.		
ID	STS_CRYPTO_00004	**State**	Draft
Affected Functional Cluster	Cryptography		
Trace to RS Criteria	[RS_CRYPTO_02002], [RS_CRYPTO_02008], [RS_CRYPTO_02202], [RS_CRYPTO_02204], [RS_CRYPTO_02205], [RS_CRYPTO_02302]		
Reference to Test Environment	STC_CRYPTO_00001 in Test configurations		
Configuration Parameters	- Allow use of asymmetric key pair <APbK> and <APvK> for generation of digital signature by [CRYPTO Tester] and [CRYPTOApp01].		
Summary	[CRYPTO Tester] sends <Data1> to [CRYPTOApp01] and digital signature <DS1'> is generated by [CRYPTOApp01] from <Data1> using [CRYPTOApp01]'s private key <APvK>. <DS1'> is compared with <DS1> which is generated in the same way on the [CRYPTO Tester] side. <Data2> and <DS2> are sent from [CRYPTO Tester] to [CRYPTOApp01] and <Data1> is verified by [CRYPTOApp01] using <DS2> and [CRYPTOApp01]'s public key <APbK>. - Generation of <DS1> and <DS2> on the [CRYPTO Tester] side is performed either prior to running test or during a test step. - Whether to compare <DS1'> in [CRYPTOApp01] or [CRYPTO Tester] is up to implementer.		
Pre-conditions	- Crypto stack and [CRYPTOApp01] are initialized with used key, algorithm, and domain parameter as applicable. - Communication between [CRYPTO Tester] and [CRYPTOApp01] has been set up.		
Post-conditions	Communication between [CRYPTO Tester] and [CRYPTOApp01] is closed.		
Main Test Execution			
Test Steps		**Pass Criteria**	
Step 1	[CRYPTO Tester] Send <Data1> to [CRYPTOApp01].		
Step 2	[CRYPTOApp01] Generate digital signature <DS1'> using <Data1> and [CRYPTOApp01]'s private key <APvK> (via HashFunctionCtx::Start()/Update()/Finish() and SignerPrivateCtx::Sign()).		
Step 3	[CRYPTOApp01] Return digital signature generation status to [CRYPTO Tester].		
Step 4	[CRYPTO Tester] Check digital signature generation status.	[CRYPTO Tester] Digital signature generation status contains success and no error.	
Step 5	[CRYPTO Tester] Send <DS1> to [CRYPTOApp01].		
Step 6	[CRYPTOApp01] Compare <DS1'> with <DS1>.		
Step 7	[CRYPTOApp01] Return comparison result (matched/unmatched) to [CRYPTO Tester].		
Step 8	[CRYPTO Tester] Check comparison result.	[CRYPTO Tester] Comparison result is "matched."	
Step 9	[CRYPTO Tester] Send <Data2> and <DS2> to [CRYPTOApp01].		

▽

△

Step 10	[CRYPTOApp01] Verify <DS2> using [CRYPTOApp01]'s public key <APbK> (via HashFunctionCtx::Start()/Update()/Finish() and VerifierPublicCtx::Verify()).	
Step 11	[CRYPTOApp01] Return <DS2> verification status to [CRYPTO Tester].	
Step 12	[CRYPTO Tester] Check <DS2> verification status.	[CRYPTO Tester] Verification status contains success and no error.

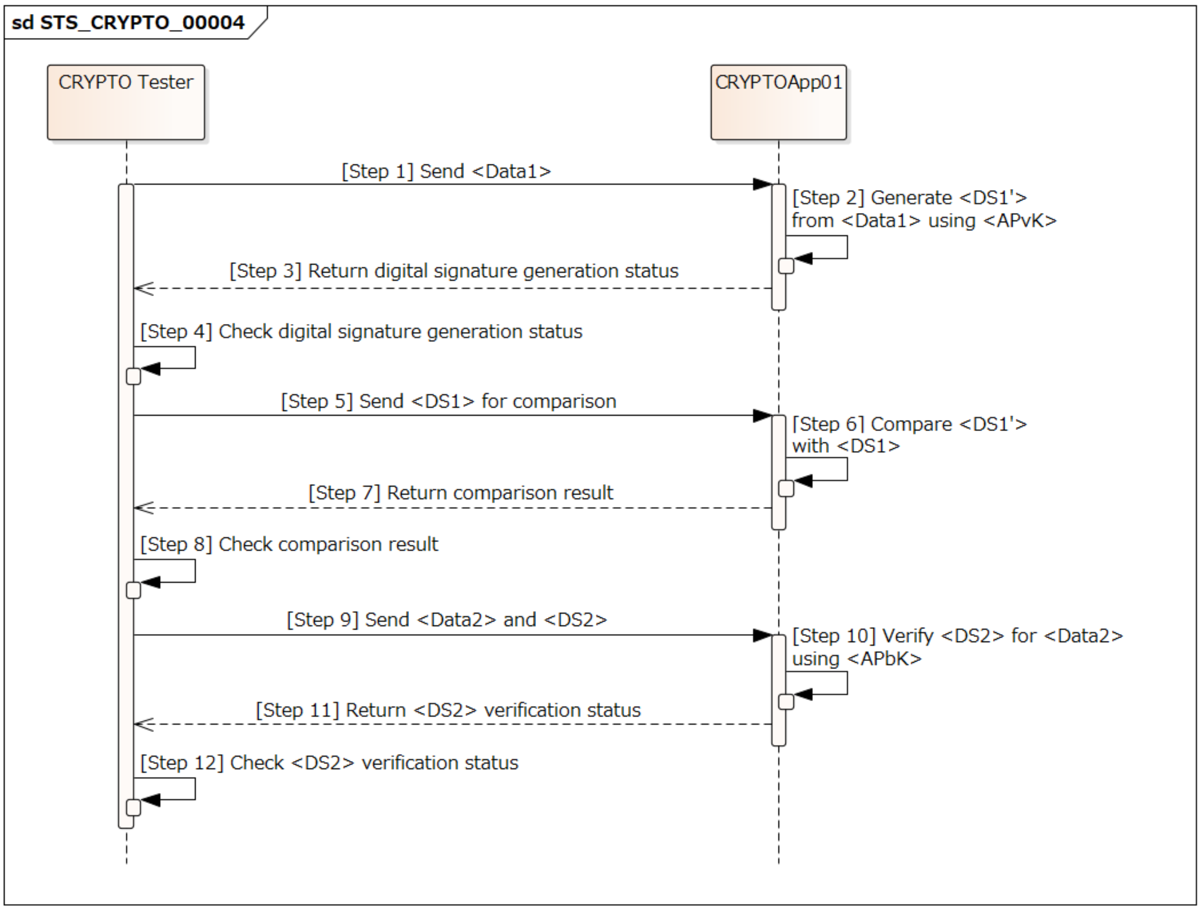
sd STS_CRYPTO_00004
CRYPTO Tester
CRYPTOApp01
[Step 1] Send <Data1>
[Step 2] Generate <DS1'> from <Data1> using <APvK>
[Step 3] Return digital signature generation status
[Step 4] Check digital signature generation status
[Step 5] Send <DS1> for comparison
[Step 6] Compare <DS1'> with <DS1>
[Step 7] Return comparison result
[Step 8] Check comparison result
[Step 9] Send <Data2> and <DS2>
[Step 10] Verify <DS2> for <Data2> using <APbK>
[Step 11] Return <DS2> verification status
[Step 12] Check <DS2> verification status

Figure 15.3: Sequence diagram of STS_CRYPTO_00004.

15.2.5 [STS_CRYPTO_00005] Generation of hash value.

Test Objective	Verify that Crypto Stack correctly generates hash value.		
ID	STS_CRYPTO_00005	**State**	Draft
Affected Functional Cluster	Cryptography		
Trace to RS Criteria	[RS_CRYPTO_02302]		
Reference to Test Environment	STC_CRYPTO_00001 in Test configurations		
Configuration Parameters	-		
Summary	[CRYPTO Tester] sends <Data1> to [CRYPTOApp01] and hash value <Hash1'> is generated by [CRYPTOApp01] from <Data1>. <Hash1'> is compared with <Hash1> which is generated in the same way on the [CRYPTO Tester] side. - Generation of <Hash1> on the [CRYPTO Tester] side is performed either prior to running test or during a test step. - Whether to compare <Hash1'> in [CRYPTOApp01] or [CRYPTO Tester] is up to implementer.		
Pre-conditions	- Crypto stack and [CRYPTOApp01] are initialized with used algorithm and domain parameter as applicable. - Communication between [CRYPTO Tester] and [CRYPTOApp01] has been set up.		
Post-conditions	Communication between [CRYPTO Tester] and [CRYPTOApp01] is closed.		
Main Test Execution			
Test Steps			**Pass Criteria**
Step 1	[CRYPTO Tester] Send <Data1> to [CRYPTOApp01].		
Step 2	[CRYPTOApp01] Generate <Hash1'> from <Data1> (via HashFunctionCtx::Start()/Update()/Finish()).		
Step 3	[CRYPTOApp01] Return hash value generation status to [CRYPTO Tester].		
Step 4	[CRYPTO Tester] Check hash value generation status.		[CRYPTO Tester] Hash value generation status contains success and no error.
Step 5	[CRYPTO Tester] Send <Hash1> to [CRYPTOApp01].		
Step 6	[CRYPTOApp01] Compare <Hash1'> with <Hash1> (via HashFunctionCtx::Compare()).		
Step 7	[CRYPTOApp01] Return comparison status to [CRYPTO Tester].		
Step 8	[CRYPTO Tester] Check comparison status.		[CRYPTO Tester] Comparison status contains success and no error.

15.2.6 [STS_CRYPTO_00006] Generation of random number.

Test Objective	Verify that Crypto Stack correctly generates random numbers.		
ID	STS_CRYPTO_00006	**State**	Draft
Affected Functional Cluster	Cryptography		
Trace to RS Criteria	[RS_CRYPTO_02206]		
Reference to Test Environment	STC_CRYPTO_00001 in Test configurations		
Configuration Parameters	-		
Summary	[CRYPTO Tester] sends <Input1> (optional) to [CRYPTOApp01] to trigger random number generation. [CRYPTOApp01] generates a random number <RN1'> and generation status is checked to have no error. [CRYPTO Tester] sends <RN1> (generated with same input and algorithm as in [CRYPTOApp01]) to [CRYPTOApp01]. [CRYPTOApp01] compares <RN1'> with <RN1> generation status and comparison result is checked to match. - <RN1> is generated in [CRYPTO Tester] either prior to running test or during a test step. - Whether to compare <RN1> and <RN1'> in [CRYPTOApp01] or [CRYPTO Tester] is up to implementer.		
Pre-conditions	- Crypto stack and [CRYPTOApp01] are initialized with used algorithm. - Communication between [CRYPTO Tester] and [CRYPTOApp01] has been set up.		
Post-conditions	Communication between [CRYPTO Tester] and [CRYPTOApp01] is closed.		
Main Test Execution			
Test Steps		**Pass Criteria**	
Step 1	[CRYPTO Tester] Send <Input1> to [CRYPTOApp01] to trigger random number generation (send e.g. 0 for <Input1> if no input is needed for used algorithm).		
Step 2	[CRYPTOApp01] Generate random number (using <Input1> as needed) to obtain <RN1'>.		
Step 3	[CRYPTOApp01] Return <RN1'> generation status (success/failure) to [CRYPTO Tester].		
Step 4	[CRYPTO Tester] Check <RN1'> generation status.	[CRYPTO Tester] <RN1'> generation status contains no error.	
Step 5	[CRYPTO Tester] Send <RN1> (generated in [CRYPTO Tester]) to [CRYPTOApp01] to trigger random number comparison.		
Step 6	[CRYPTOApp01] Compare random numbers <RN1'> with <RN1>.		
Step 7	[CRYPTOApp01] Return comparison result (matched/unmatched) to [CRYPTO Tester].		
Step 8	[CRYPTO Tester] Check comparison result.	[CRYPTO Tester] Comparison result is "matched."	

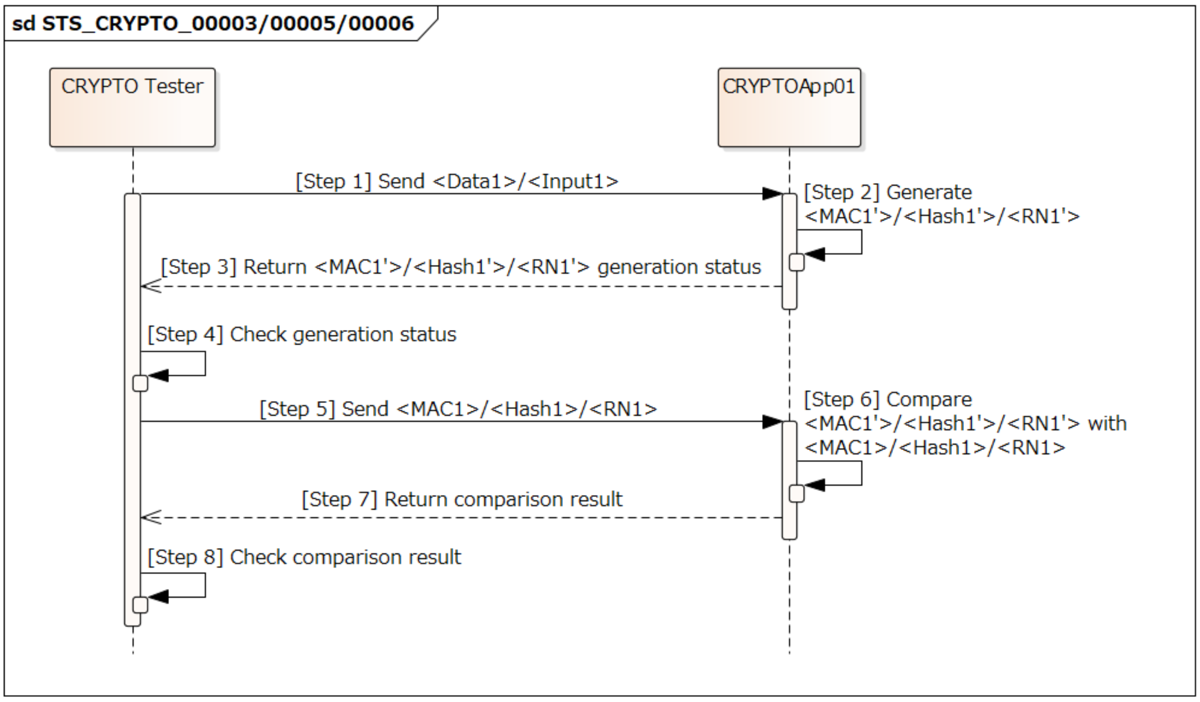
sd STS_CRYPTO_00003/00005/00006
CRYPTO Tester
CRYPTOApp01
[Step 1] Send <Data1>/<Input1>
[Step 2] Generate <MAC1'>/<Hash1'>/<RN1'>
[Step 3] Return <MAC1'>/<Hash1'>/<RN1'> generation status
[Step 4] Check generation status
[Step 5] Send <MAC1>/<Hash1>/<RN1>
[Step 6] Compare <MAC1'>/<Hash1'>/<RN1'> with <MAC1>/<Hash1>/<RN1>
[Step 7] Return comparison result
[Step 8] Check comparison result

Figure 15.4: Sequence diagram of STS_CRYPTO_00003/00005/00006.

15.2.7 [STS_CRYPTO_00007] Authenticated symmetric encryption and decryption.

Test Objective	Verify that Crypto Stack correctly performs authenticated encryption and decryption.		
ID	STS_CRYPTO_00007	**State**	Draft
Affected Functional Cluster	Cryptography		
Trace to RS Criteria	[RS_CRYPTO_02001], [RS_CRYPTO_02008], [RS_CRYPTO_02201], [RS_CRYPTO_02207], [RS_CRYPTO_02302]		
Reference to Test Environment	STC_CRYPTO_00001 in Test configurations		
Configuration Parameters	- Configure [CRYPTOApp01] to allow use of symmetric key for authenticated symmetric encryption/decryption algorithm.		
Summary	[CRYPTO Tester] sends plaintext <Plaintext1> and optionally associated data <ASData1> to [CRYPTOApp01] to test generation of authenticated ciphertext (AC). [CRYPTOApp01] generates authenticated ciphertext <AC1'> consists of encrypted <Plaintext1>, optionally <ASData1>, and message authentication code (MAC). <AC1'> is compared with <AC1> generated by [CRYPTO Tester]. [CRYPTO Tester] generates <AC2> from <Plaintext2> and optionally <ASData2> and sends <AC2> to [CRYPTOApp01] for decryption. [CRYPTOApp01] decrypts <AC2> to obtain <Plaintext2'>, <MAC2'>, and optionally <ASData2'>, which are checked for correctness. ▽		

▽

△

	△ - <AC1> and <AC2> are generated on the [CRYPTO Tester] side either prior to running test or during test steps. - Whether to compare <AC1> and <Plaintext2> in [CRYPTOApp01] or [CRYPTO Tester] is up to implementer.	
Pre-conditions	- Crypto stack and [CRYPTOApp01] are initialized with used key, algorithm, and domain parameter as applicable. - Communication between [CRYPTO Tester] and [CRYPTOApp01] has been set up. - A symmetric key is shared between [CRYPTO Tester] and [CRYPTOApp01] for encryption and decryption of <AC1> and <AC2>.	
Post-conditions	Communication between [CRYPTO Tester] and [CRYPTOApp01] is closed.	
Main Test Execution		
Test Steps		**Pass Criteria**
Step 1	[CRYPTO Tester] Send <Plaintext1> and optionally <ASData1> to trigger <AC1'> generation.	
Step 2	[CRYPTOApp01] Generate <AC1'> from <Plaintext1> and optionally <ASData1>.	
Step 3	[CRYPTOApp01] Return <AC1'> generation status to [CRYPTO Tester].	
Step 4	[CRYPTO Tester] Check <AC1'> generation status.	[CRYPTO Tester] <AC1'> generation status contains no error.
Step 5	[CRYPTO Tester] Send <AC1> to [CRYPTOApp01] for comparison.	
Step 6	[CRYPTOApp01] Compare <AC1'> with <AC1>.	
Step 7	[CRYPTOApp01] Return <AC1> comparison result (matched/unmatched) to [CRYPTO Tester].	
Step 8	[CRYPTO Tester] Check <AC1> comparison result.	[CRYPTO Tester] Comparison result is "matched."
Step 9	[CRYPTO Tester] Send <AC2> to [CRYPTOApp01] to trigger decryption.	
Step 10	[CRYPTOApp01] Decrypt <AC2> to obtain <Plaintext2'>, <MAC2'> and optionally <ASData2'>.	
Step 11	[CRYPTOApp01] Return <AC2> decryption status to [CRYPTO Tester].	
Step 12	[CRYPTO Tester] Check <AC2> decryption status.	[CRYPTO Tester] Decryption status contains no error.
Step 13	[CRYPTO Tester] Send <Plaintext2> and optionally <ASData2> to [CRYPTOApp01] for comparison.	

▽

△

Step 14	[CRYPTOApp01] Compare <Plaintext2'> with <Plaintext2> and <ASData2'> with <ASData2>.	
Step 15	[CRYPTOApp01] Return comparison result (matched/unmatched) to [CRYPTO Tester].	
Step 16	[CRYPTO Tester] Check comparison result.	[CRYPTO Tester] Comparison result is "matched."
Step 17	[CRYPTO Tester] Send trigger of <MAC2'> verification to [CRYPTOApp01].	
Step 18	[CRYPTOApp01] Verify <MAC2'> of <AC2>.	
Step 19	[CRYPTOApp01] Return <MAC2'> verification result (matched/unmatched) to [CRYPTO Tester].	
Step 20	[CRYPTO Tester] Check <MAC2'> verification result.	[CRYPTO Tester] Verification result is "matched."

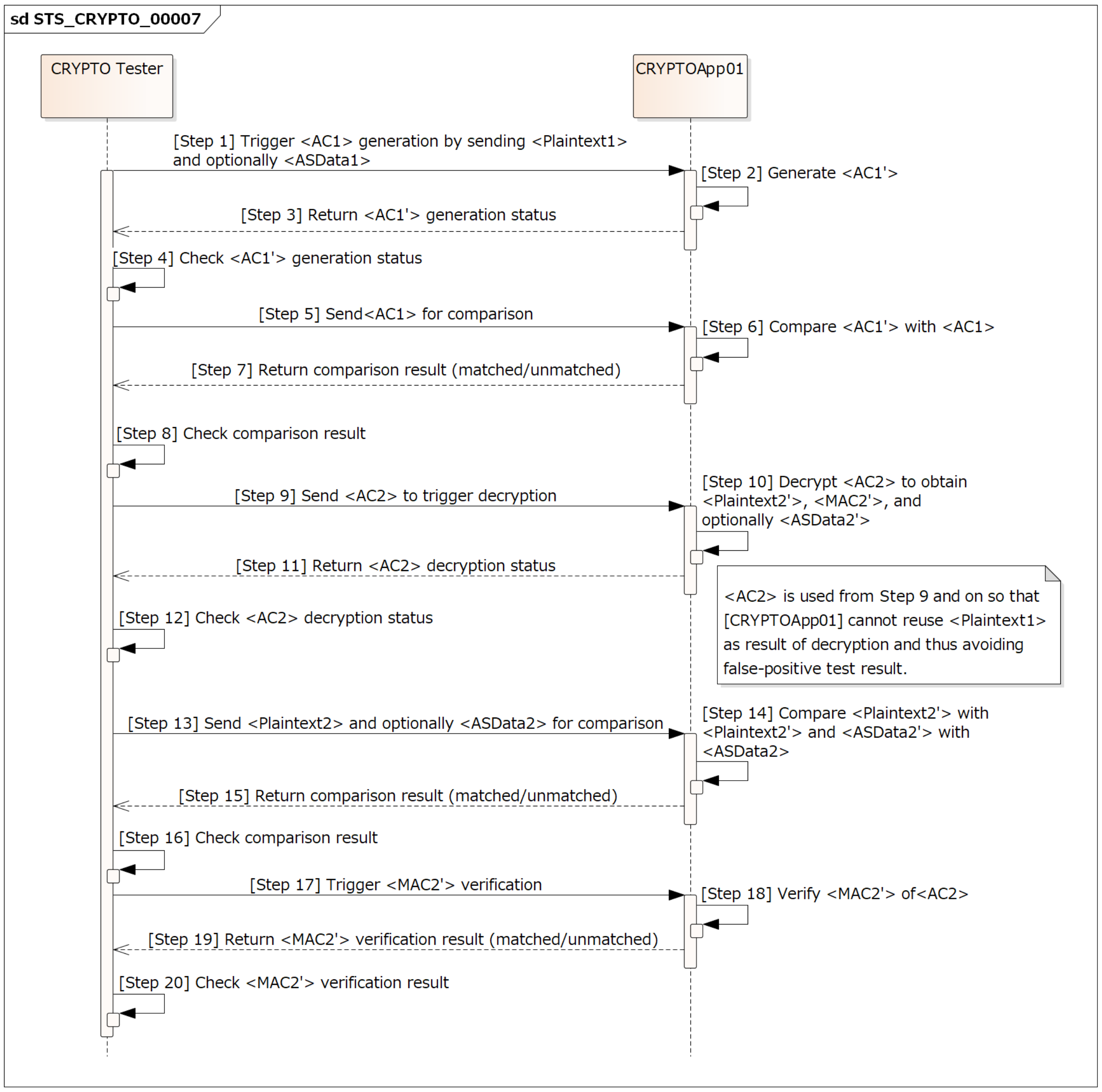
sd STS_CRYPTO_00007
CRYPTO Tester
CRYPTOApp01
[Step 1] Trigger <AC1> generation by sending <Plaintext1> and optionally <ASData1>
[Step 2] Generate <AC1'>
[Step 3] Return <AC1'> generation status
[Step 4] Check <AC1'> generation status
[Step 5] Send<AC1> for comparison
[Step 6] Compare <AC1'> with <AC1>
[Step 7] Return comparison result (matched/unmatched)
[Step 8] Check comparison result
[Step 9] Send <AC2> to trigger decryption
[Step 10] Decrypt <AC2> to obtain <Plaintext2'>, <MAC2'>, and optionally <ASData2'>
[Step 11] Return <AC2> decryption status
<AC2> is used from Step 9 and on so that [CRYPTOApp01] cannot reuse <Plaintext1> as result of decryption and thus avoiding false-positive test result.
[Step 12] Check <AC2> decryption status
[Step 13] Send <Plaintext2> and optionally <ASData2> for comparison
[Step 14] Compare <Plaintext2'> with <Plaintext2'> and <ASData2'> with <ASData2>
[Step 15] Return comparison result (matched/unmatched)
[Step 16] Check comparison result
[Step 17] Trigger <MAC2'> verification
[Step 18] Verify <MAC2'> of<AC2>
[Step 19] Return <MAC2'> verification result (matched/unmatched)
[Step 20] Check <MAC2'> verification result

Figure 15.5: Sequence diagram of STS_CRYPTO_00007.

15.2.8 [STS_CRYPTO_00008] Key wrapping/unwrapping and key encapsulation/decapsulation.

Test Objective	Verify that Crypto Stack correctly performs key encapsulation/decapsulation, together with key wrapping/unwrapping.		
ID	STS_CRYPTO_00008	**State**	Draft
Affected Functional Cluster	Cryptography		

▽

△

Trace to RS Criteria	[RS_CRYPTO_02001], [RS_CRYPTO_02002], [RS_CRYPTO_02008], [RS_CRYPTO_02104], [RS_CRYPTO_02201], [RS_CRYPTO_02202], [RS_CRYPTO_02208], [RS_CRYPTO_02209]	
Reference to Test Environment	STC_CRYPTO_00001 in Test configurations	
Configuration Parameters	- Configure [CRYPTO Tester] to have symmetric keys <SK1> and <SK2> for key wrapping/unwrapping algorithm. - Configure [CRYPTO Tester] to allow use of its asymmetric key pair: public key <TPbK> and private key <TPvK>, and [CRYPTOApp01]'s public key <APbK> for key encapsulation/decapsulation algorithm. - Configure [CRYPTOApp01] to allow use of its asymmetric key pair: public key <APbK> and private key <APvK>, and [CRYPTO Tester]'s public key <TPbK> for key encapsulation/decapsulation algorithm.	
Summary	[CRYPTO Tester] sends an encapsulated key to [CRYPTOApp01] to trigger decapsulation of the key. [CRYPTOApp01] decapsulates the key and returns the decapsulation status to [CRYPTO Tester] for checking. [CRYPTO Tester] sends a plaintext data to test whether decapsulated key on the [CRYPTOApp01] works correctly. [CRYPTO Tester] triggers to [CRYPTOApp01] for key encapsulation. [CRYPTO App01] encapsulates a symmetric key and returns the encapsulation status to [CRYPTO Tester] for checking. Encapsulated key on the [CRYPTOApp01] side is checked by comparing with one created in the same way on the [CRYPTO Tester] side. The above is performed also for key wrapping/unwrapping. - Key encapsulation/decapsulation and wrapping/unwrapping on the [CRYPTO Tester] side are done either prior to running test or during test steps - Whether to compare result data (e.g. <Ciphertext1> and <Ciphertext1'>) in [CRYPTOApp01] or [CRYPTO Tester] is up to implementer.	
Pre-conditions	- [CRYPTO Tester] has an encapsulated symmetric key <ESK1_APbK> (symmetric key <SK1>, encapsulated with [CRYPTOApp01]'s public key <APbK>). - [CRYPTO Tester] has a wrapped symmetric key <WSK2> (symmetric key <SK2> wrapped by <SK1>). - Crypto stack and [CRYPTOApp01] are initialized with used key, algorithm, and domain parameter as applicable. - Communication between [CRYPTO Tester] and [CRYPTOApp01] has been set up.	
Post-conditions	Communication between [CRYPTO Tester] and [CRYPTOApp01] is closed.	
Main Test Execution		
Test Steps		**Pass Criteria**
Step 1	[CRYPTO Tester] Send <ESK1_APbK> to [CRYPTOApp01] to trigger key decapsulation.	
Step 2	[CRYPTOApp01] Decapsulate <ESK1_APbK> using its private key <APvK> to obtain <SK1>.	
Step 3	[CRYPTOApp01] Return key decapsulation status to [CRYPTO Tester].	
Step 4	[CRYPTO Tester] Check key decapsulation status.	[CRYPTO Tester] Key decapsulation status contains success and no error.
Step 5	[CRYPTO Tester] Send <Plaintext1> to [CRYPTOApp01].	

▽

△

Step 6	[CRYPTOApp01] Encrypt <Plaintext1> using <SK1> (obtained in Step 2) to obtain <Ciphertext1'>.	
Step 7	[CRYPTOApp01] Return encryption status to [CRYPTO Tester].	
Step 8	[CRYPTO Tester] Check encryption status.	[CRYPTO Tester] Encryption status contains success and no error.
Step 9	[CRYPTO Tester] Send <Ciphertext1> (encrypted <Plaintext1> using <SK1>) to [CRYPTOApp01] for comparison.	
Step 10	[CRYPTOApp01] Compare <Ciphertext1'> with <Ciphertext1>.	
Step 11	[CRYPTOApp01] Return comparison result (matched/unmatched) to [CRYPTO Tester].	
Step 12	[CRYPTO Tester] Check comparison result.	[CRYPTO Tester] Comparison result is "matched."
Step 13	[CRYPTO Tester] Trigger encapsulation of <SK1> to [CRYPTOApp01].	
Step 14	[CRYPTOApp01] Encapsulate <SK1> using <TPbK> to obtain <ESK1_TPbK'>.	
Step 15	[CRYPTOApp01] Return <SK1> encapsulation status to [CRYPTO Tester].	
Step 16	[CRYPTO Tester] Check key encapsulation status.	[CRYPTO Tester] Key encapsulation status contains success and no error.
Step 17	[CRYPTO Tester] Send <ESK1_TPbK> (encapsulated <SK1> by public key <TPbK>) to [CRYPTOApp01] for comparison.	
Step 18	[CRYPTOApp01] Compare <ESK1_TPbK'> with <ESK1_TPbK>.	
Step 19	[CRYPTOApp01] Return comparison result (matched/unmatched) to [CRYPTO Tester].	
Step 20	[CRYPTO Tester] Check comparison result.	[CRYPTO Tester] Comparison result is "matched."
Step 21	[CRYPTO Tester] Send <WSK2> to [CRYPTOApp01] to trigger key unwrapping.	
Step 22	[CRYPTOApp01] Unwrap <WSK2> using <SK1> to obtain <SK2>.	

▽

△

Step 23	[CRYPTOApp01] Return key unwrapping status to [CRYPTO Tester].	
Step 24	[CRYPTO Tester] Check key unwrapping status.	[CRYPTO Tester] Key unwrapping status contains success and no error.
Step 25	[CRYPTO Tester] Send <Plaintext2> to [CRYPTOApp01].	
Step 26	[CRYPTOApp01] Encrypt <Plaintext2> using <SK2> (obtained in Step 22) to obtain <Ciphertext2'>.	
Step 27	[CRYPTOApp01] Return <Plaintext2> encryption status to [CRYPTO Tester].	
Step 28	[CRYPTO Tester] Check encryption status.	[CRYPTO Tester] Encryption status contains success and no error.
Step 29	[CRYPTO Tester] Send <Ciphertext2> (encrypted <Plaintext2> using <SK2>) to [CRYPTOApp01] for comparison.	
Step 30	[CRYPTOApp01] Compare <Ciphertext2'> with <Ciphertext2>.	
Step 31	[CRYPTOApp01] Return comparison result (matched/unmatched) to [CRYPTO Tester].	
Step 32	[CRYPTO Tester] Check comparison result.	[CRYPTO Tester] Comparison result is "matched."
Step 33	[CRYPTO Tester] Trigger wrapping of <SK2> to [CRYPTOApp01].	
Step 34	[CRYPTOApp01] Wrap <SK2> using <SK1> to obtain <WSK2'>.	
Step 35	[CRYPTOApp01] Return <SK2> wrapping status to [CRYPTO Tester].	
Step 36	[CRYPTO Tester] Check key wrapping status.	[CRYPTO Tester] Key wrapping status contains success and no error.
Step 37	[CRYPTO Tester] Send trigger to [CRYPTOApp01] for <WSK2> comparison.	
Step 38	[CRYPTOApp01] Compare <WSK2'> with <WSK2>.	
Step 39	[CRYPTOApp01] Return comparison result (matched/unmatched) to [CRYPTO Tester].	
Step 40	[CRYPTO Tester] Check comparison result.	[CRYPTO Tester] Comparison result is "matched."

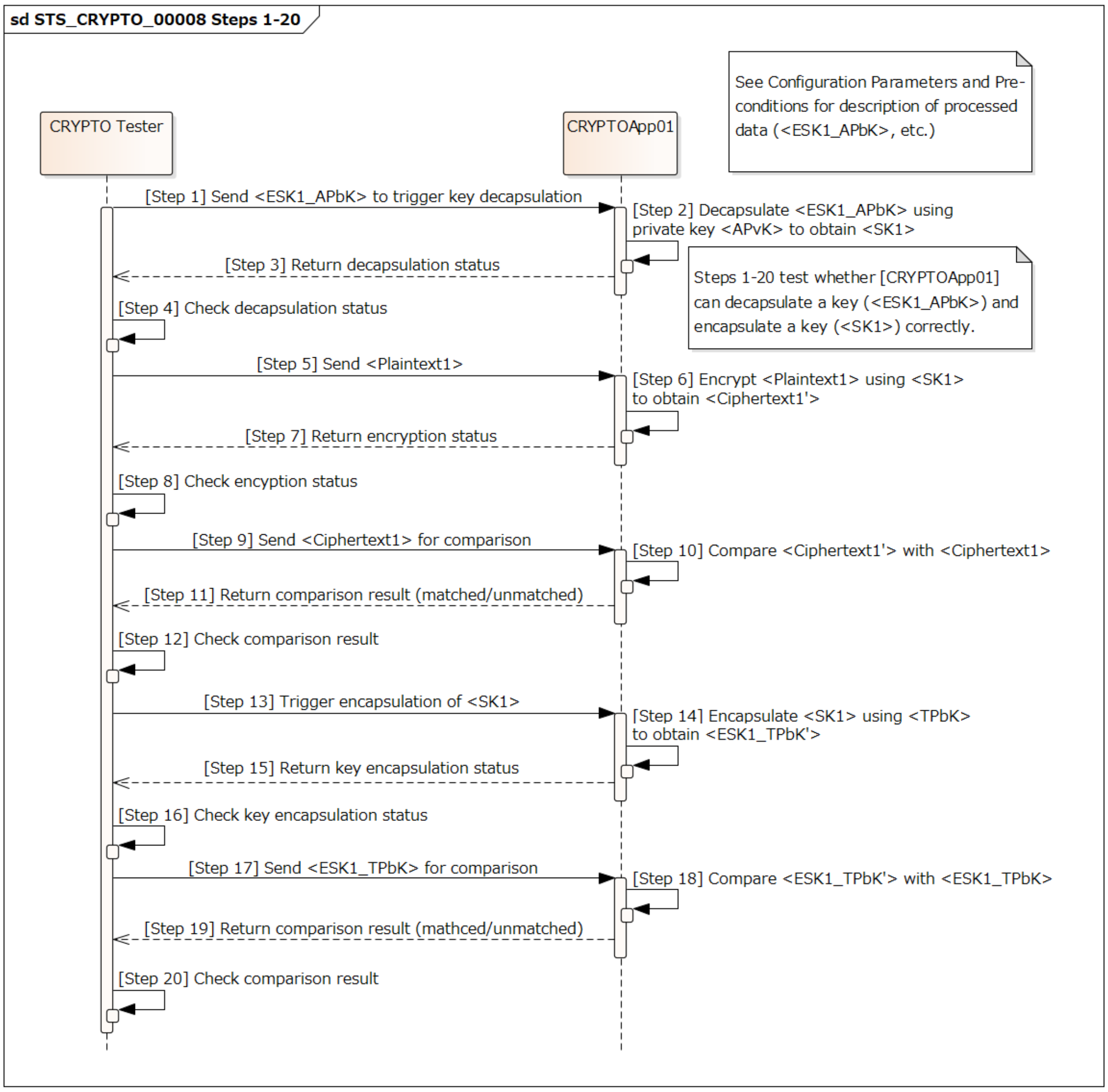
sd STS_CRYPTO_00008 Steps 1-20
See Configuration Parameters and Pre-conditions for description of processed data (<ESK1_APbK>, etc.)
CRYPTO Tester
CRYPTOApp01
[Step 1] Send <ESK1_APbK> to trigger key decapsulation
[Step 2] Decapsulate <ESK1_APbK> using private key <APvK> to obtain <SK1>
[Step 3] Return decapsulation status
Steps 1-20 test whether [CRYPTOApp01] can decapsulate a key (<ESK1_APbK>) and encapsulate a key (<SK1>) correctly.
[Step 4] Check decapsulation status
[Step 5] Send <Plaintext1>
[Step 6] Encrypt <Plaintext1> using <SK1> to obtain <Ciphertext1'>
[Step 7] Return encryption status
[Step 8] Check encyption status
[Step 9] Send <Ciphertext1> for comparison
[Step 10] Compare <Ciphertext1'> with <Ciphertext1>
[Step 11] Return comparison result (matched/unmatched)
[Step 12] Check comparison result
[Step 13] Trigger encapsulation of <SK1>
[Step 14] Encapsulate <SK1> using <TPbK> to obtain <ESK1_TPbK'>
[Step 15] Return key encapsulation status
[Step 16] Check key encapsulation status
[Step 17] Send <ESK1_TPbK> for comparison
[Step 18] Compare <ESK1_TPbK'> with <ESK1_TPbK>
[Step 19] Return comparison result (mathced/unmatched)
[Step 20] Check comparison result

Figure 15.6: Sequence diagram of STS_CRYPTO_00008 Steps 1-20.

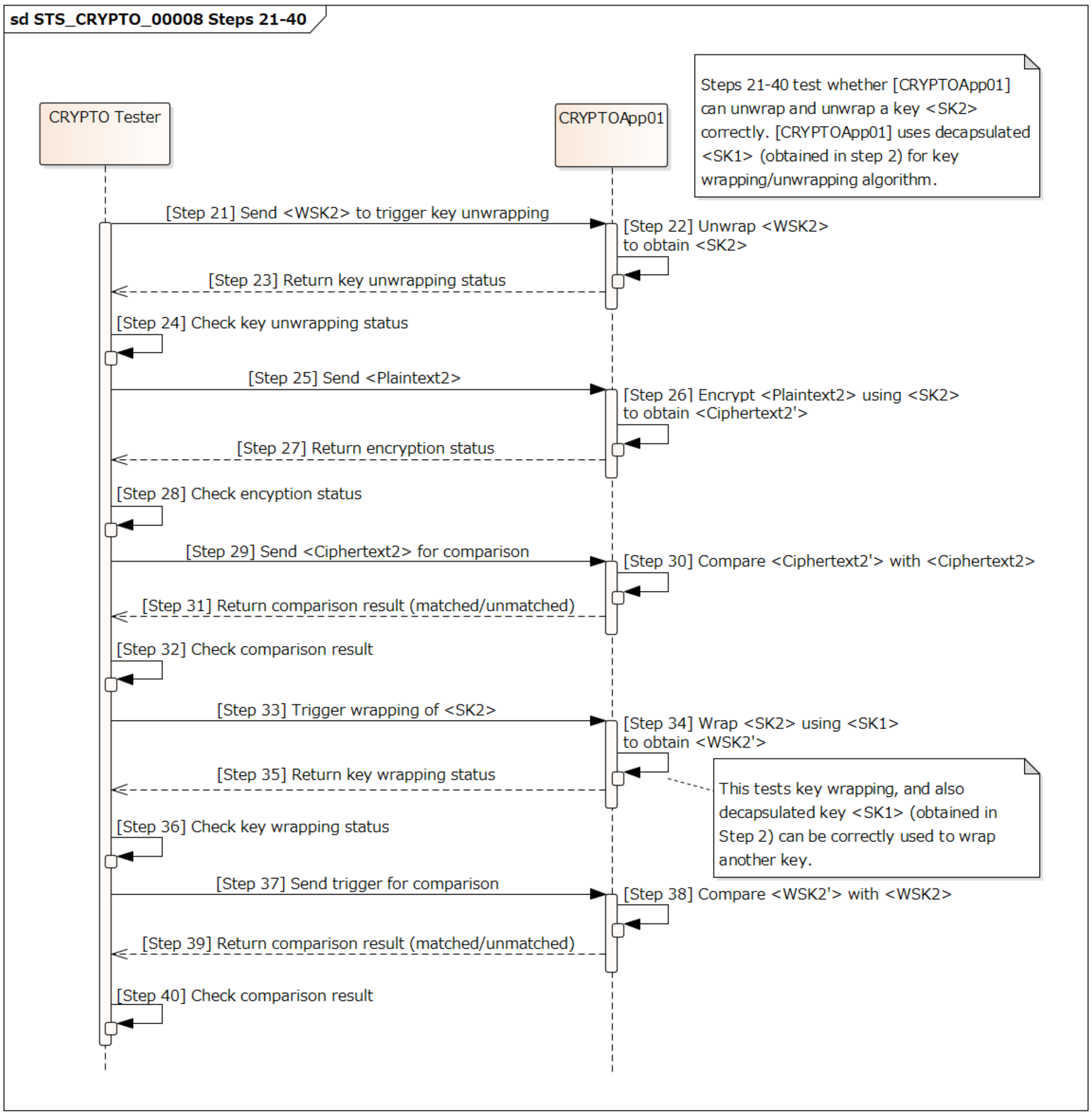
sd STS_CRYPTO_00008 Steps 21-40
CRYPTO Tester
CRYPTOApp01
Steps 21-40 test whether [CRYPTOApp01] can unwrap and unwrap a key <SK2> correctly. [CRYPTOApp01] uses decapsulated <SK1> (obtained in step 2) for key wrapping/unwrapping algorithm.
[Step 21] Send <WSK2> to trigger key unwrapping
[Step 22] Unwrap <WSK2> to obtain <SK2>
[Step 23] Return key unwrapping status
[Step 24] Check key unwrapping status
[Step 25] Send <Plaintext2>
[Step 26] Encrypt <Plaintext2> using <SK2> to obtain <Ciphertext2'>
[Step 27] Return encryption status
[Step 28] Check encyption status
[Step 29] Send <Ciphertext2> for comparison
[Step 30] Compare <Ciphertext2'> with <Ciphertext2>
[Step 31] Return comparison result (matched/unmatched)
[Step 32] Check comparison result
[Step 33] Trigger wrapping of <SK2>
[Step 34] Wrap <SK2> using <SK1> to obtain <WSK2'>
[Step 35] Return key wrapping status
This tests key wrapping, and also decapsulated key <SK1> (obtained in Step 2) can be correctly used to wrap another key.
[Step 36] Check key wrapping status
[Step 37] Send trigger for comparison
[Step 38] Compare <WSK2'> with <WSK2>
[Step 39] Return comparison result (matched/unmatched)
[Step 40] Check comparison result

Figure 15.7: Sequence diagram of STS_CRYPTO_00008 Steps 21-40.

15.2.9 [STS_CRYPTO_00009] Restriction of the allowed usage scope for keys and secret seeds.

Test Objective	Verify that Crypto Stack correctly restricts the allowed usage scope for a keys and secret seeds.		
ID	STS_CRYPTO_00009	**State**	Draft
Affected Functional Cluster	Cryptography		

▽

△

Trace to RS Criteria	[RS_CRYPTO_02008]	
Reference to Test Environment	STC_CRYPTO_00001 in Test configurations	
Configuration Parameters	- Configure [CRYPTO Tester] to have a key <Key1> or secret seed <Seed1> with allowed usage <Usage1>. - Configure [CRYPTOApp01] to have <Key1> or <Seed1> with allowed usage <Usage1> (same as CRYPTO Tester).	
Summary	[CRYPTO Tester] checks whether [CRYPTOApp01] can retrieve allowed usage information of configured <Key1> or <Seed1>, by comparing expected <AllowedUsageFlags1> and <AllowedUsageFlags1'> retrieved by [CRYPTOApp01] via CryptoAPI. [CRYPTO Tester] checks whether <Key1> or <Seed1> can only be used for allowed usage <Usage1>, by triggering allowed usage <Usage1> and comparing the resulting data <Result1>, and by triggering disallowed usage <Usage2> expecting failure. - Used algorithms and values for <Key1>, <Seed1>, <AllowedUsageFlags1>, <Usage1>, and <Usage2> are chosen so that the test can be performed. - Execution of <Usage1> using <Key1> or <Seed1> (e.g. encryption, key derivation, etc.) on the [CRYPTO Tester] side is performed either prior to running test or during a test step. - Whether to compare <AllowedUsageFlags1> and <Result1> in [CRYPTOApp01] or [CRYPTO Tester] is up to implementer.	
Pre-conditions	- [CRYPTO Tester] is initialized with configured (expected) allowed usage information <AllowedUsageFlags1> of <Key1> or <Seed1> for [CRYPTOApp01]. - Crypto stack and [CRYPTOApp01] are initialized with <Key1> or <Seed1>, algorithm, and domain parameter as applicable. - Communication between [CRYPTO Tester] and [CRYPTOApp01] has been set up.	
Post-conditions	Communication between [CRYPTO Tester] and [CRYPTOApp01] is closed.	
Main Test Execution		
Test Steps		**Pass Criteria**
Step 1	[CRYPTO Tester] Send trigger of allowed usage retrieval to [CRYPTOApp01].	
Step 2	[CRYPTOApp01] Retrieve <AllowedUsageFlags1'> of <Key1> or <Seed1> via CryptoAPI AllowedUsage().	
Step 3	[CRYPTOApp01] Return <AllowedUsageFlags1'> to [CRYPTO Tester].	
Step 4	[CRYPTO Tester] Compare <AllowedUsageFlags1'> with <AllowedUsageFlags1> (expected value from the configuration).	[CRYPTO Tester] Comparison result is "matched."
Step 5	[CRYPTO Tester] Send trigger of executing an allowed usage <Usage1> of <Key1> or <Seed1> (e.g. encryption, key derivation, etc.) to [CRYPTOApp01], with input data as needed.	
Step 6	[CRYPTOApp01] Execute <Usage1> using <Key1> or <Seed1> to obtain <Result1'>.	
Step 7	[CRYPTOApp01] Return <Usage1> execution status (success/failure) to [CRYPTO Tester].	

▽

△

Step 8	[CRYPTO Tester] Check <Usage1> execution status.	[CRYPTO Tester] Execution status contains success and no error.
Step 9	[CRYPTO Tester] Send resulting data <Result1> of <Usage1> (e.g. send <Ciphertext1> if <Usage1> was encryption)	
Step 10	[CRYPTOApp01] Compare <Result1'> with <Result1>.	
Step 11	[CRYPTOApp01] Return comparison result (matched/unmatched) to [CRYPTO Tester].	
Step 12	[CRYPTO Tester] Check comparison result.	[CRYPTO Tester] Comparison result is "matched."
Step 13	[CRYPTO Tester] Trigger a disallowed usage <Usage2> of <Key1> or <Seed1>, with input data as needed.	
Step 14	[CRYPTOApp01] Execute disallowed usage <Usage2> using <Key1> or <Seed1>.	
Step 15	[CRYPTOApp01] Return disallowed usage <Usage2> execution status to [CRYPTO Tester].	
Step 16	[CRYPTO Tester] Check execution status.	[CRYPTO Tester] Execution status contains "kUsageViolation" error.

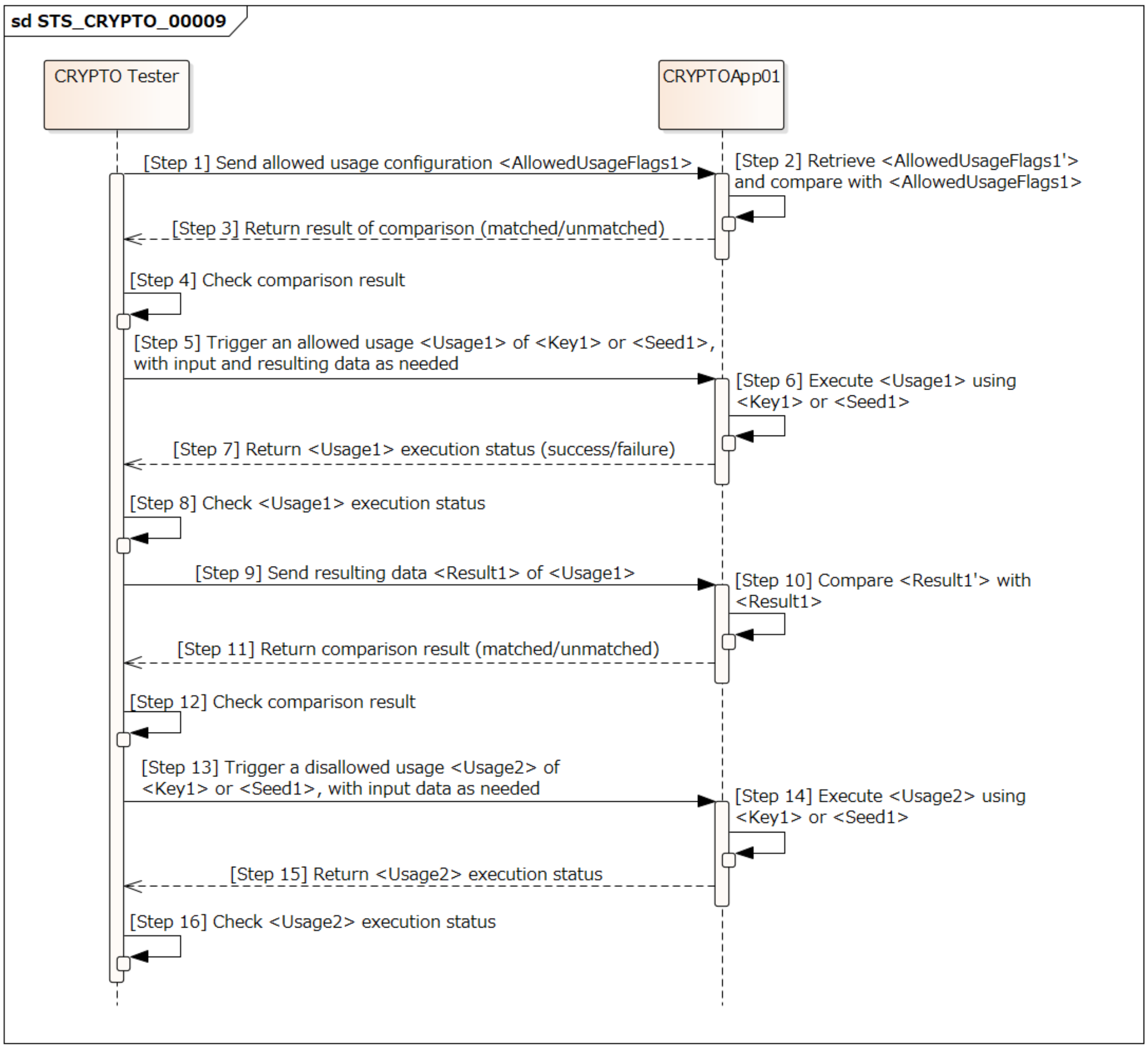
sd STS_CRYPTO_00009
CRYPTO Tester
CRYPTOApp01
[Step 1] Send allowed usage configuration <AllowedUsageFlags1>
[Step 2] Retrieve <AllowedUsageFlags1'> and compare with <AllowedUsageFlags1>
[Step 3] Return result of comparison (matched/unmatched)
[Step 4] Check comparison result
[Step 5] Trigger an allowed usage <Usage1> of <Key1> or <Seed1>, with input and resulting data as needed
[Step 6] Execute <Usage1> using <Key1> or <Seed1>
[Step 7] Return <Usage1> execution status (success/failure)
[Step 8] Check <Usage1> execution status
[Step 9] Send resulting data <Result1> of <Usage1>
[Step 10] Compare <Result1'> with <Result1>
[Step 11] Return comparison result (matched/unmatched)
[Step 12] Check comparison result
[Step 13] Trigger a disallowed usage <Usage2> of <Key1> or <Seed1>, with input data as needed
[Step 14] Execute <Usage2> using <Key1> or <Seed1>
[Step 15] Return <Usage2> execution status
[Step 16] Check <Usage2> execution status

Figure 15.8: Sequence diagram of STS_CRYPTO_00009.

15.2.10 [STS_CRYPTO_00010] Exchange of symmetric keys by Diffie-Hellman(DH)/Elliptic Curve DH(ECDH) key agreement.

Test Objective	Verify that Crypto Stack correctly exchanges symmetric key by DH/ECDH key agreement.		
ID	STS_CRYPTO_00010	**State**	Draft
Affected Functional Cluster	Cryptography		
Trace to RS Criteria	[RS_CRYPTO_02104]		
Reference to Test Environment	STC_CRYPTO_00001 in Test configurations		

▽

△

Configuration Parameters	- Configure [CRYPTO Tester] to have a public key for DH/ECDH <ADHPbK1> (as if already received from [CRYPTOApp01]). - Configure [CRYPTOApp01] to have a public key for DH/ECDH <TDHPbK1> (as if already received from [CRYPTO Tester]).	
Summary	[CRYPTO Tester] checks whether [CRYPTOApp01] correctly generates symmetric key <SK1> by calling AgreeKey() API. Generated <SK1> is checked by executing an allowed usage <Usage1> of <SK1> (e.g. encryption) in [CRYPTOApp01], checking execution status of <Usage1>, and comparing the result <Result1>. - Key agreement on the [CRYPTO Tester] side is performed either prior to running test or during a test step. - Whether to compare <Result1> in [CRYPTOApp01] or [CRYPTO Tester] is up to implementer.	
Pre-conditions	- Exchange of public keys for DH/ECDH is already done between [CRYPTO Tester] and [CRYPTOApp01]. - Crypto stack and [CRYPTOApp01] are initialized with key, algorithm, and domain parameter as applicable. - Communication between [CRYPTO Tester] and [CRYPTOApp01] has been set up.	
Post-conditions	Communication between [CRYPTO Tester] and [CRYPTOApp01] is closed.	
Main Test Execution		
Test Steps		**Pass Criteria**
Step 1	[CRYPTO Tester] Trigger DH/ECDH key agreement.	
Step 2	[CRYPTOApp01] Call AgreeKey() API using <TDHPbK1> to obtain symmetric key <SK1>.	
Step 3	[CRYPTOApp01] Return key agreement status (success/failure) to [CRYPTO Tester].	
Step 4	[CRYPTO Tester] Check key agreement status.	[CRYPTO Tester] Key agreement status contains no error.
Step 5	[CRYPTO Tester] Trigger an allowed usage <Usage1> of <SK1> (e.g. encryption) to [CRYPTOApp01] (send input data as needed).	
Step 6	[CRYPTOApp01] Execute <Usage1> using <SK1> to obtain <Result1'>.	
Step 7	[CRYPTOApp01] Return execution status(success/failure) to [CRYPTOTester].	
Step 8	[CRYPTO Tester] Check execution status.	[CRYPTO Tester] Execution status contains success and no error.
Step 9	[CRYPTO Tester] Send <Result1> (generated on the [CRYPTO Tester] side in the same way as <Result1'>) to [CRYPTOApp01] for comparison.	
Step 10	[CRYPTOApp01] Compare <Result1'> with <Result1>.	

▽

△

Step 11	[CRYPTOApp01] Return comparison result (matched/unmatched) to [CRYPTO Tester].	
Step 12	[CRYPTO Tester] Check comparison result.	[CRYPTO Tester] Comparison result is "matched."

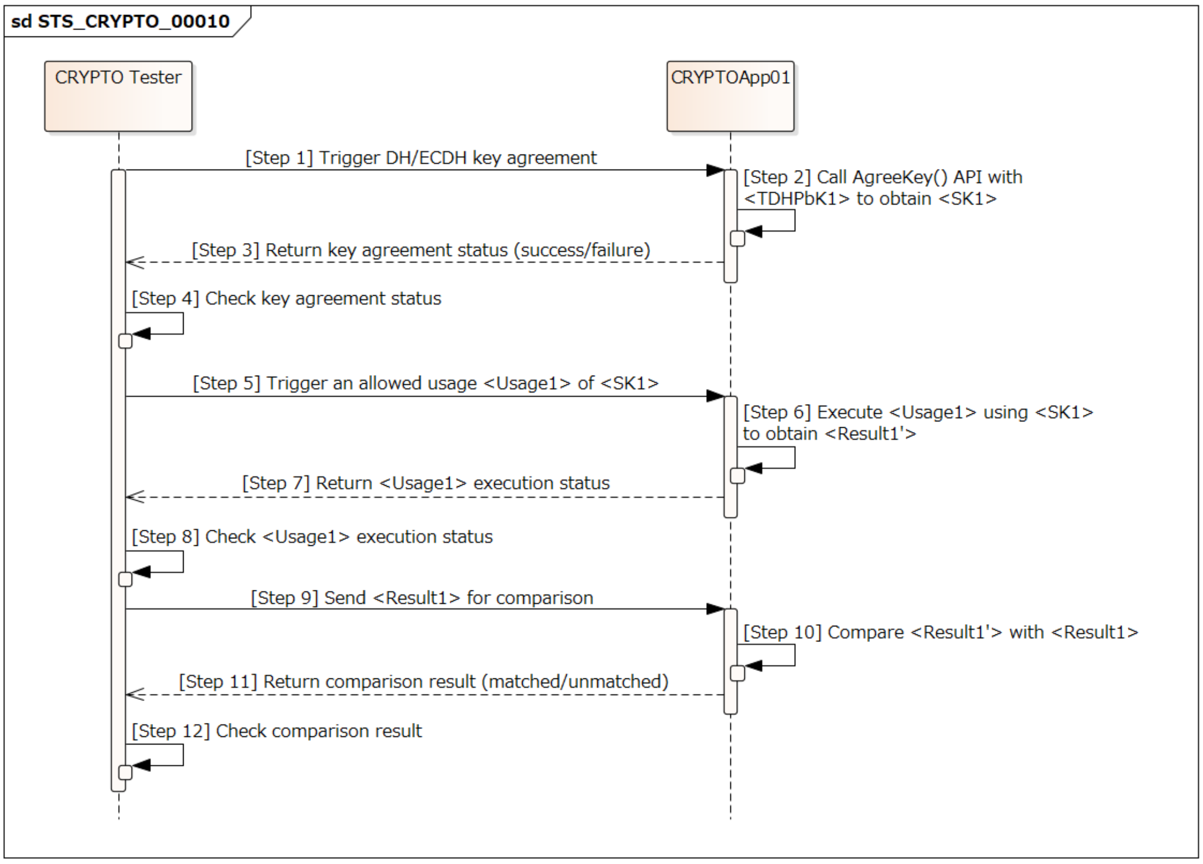
sd STS_CRYPTO_00010
CRYPTO Tester
CRYPTOApp01
[Step 1] Trigger DH/ECDH key agreement
[Step 2] Call AgreeKey() API with <TDHPbK1> to obtain <SK1>
[Step 3] Return key agreement status (success/failure)
[Step 4] Check key agreement status
[Step 5] Trigger an allowed usage <Usage1> of <SK1>
[Step 6] Execute <Usage1> using <SK1> to obtain <Result1'>
[Step 7] Return <Usage1> execution status
[Step 8] Check <Usage1> execution status
[Step 9] Send <Result1> for comparison
[Step 10] Compare <Result1'> with <Result1>
[Step 11] Return comparison result (matched/unmatched)
[Step 12] Check comparison result

Figure 15.9: Sequence diagram of STS_CRYPTO_00010.

15.2.11 [STS_CRYPTO_00011] Import and export of keys and secret seeds.

Test Objective	Verify that Crypto Stack correctly imports and exports keys and secret seeds.		
ID	STS_CRYPTO_00011	**State**	Draft
Affected Functional Cluster	Cryptography		
Trace to RS Criteria	[RS_CRYPTO_02105], [RS_CRYPTO_02112], [RS_CRYPTO_02113], [RS_CRYPTO_02115], [RS_CRYPTO_02102], [RS_CRYPTO_02007]		
Reference to Test Environment	STC_CRYPTO_00001 in Test configurations		

▽

△

Configuration Parameters	- Configure [CRYPTO Tester] to have a tested key material – either <SecureKM1> as symmetric key/asymmetric private key/secret seed, or <UnsecureKey1> as asymmetric public key – with at least one of the following meta information: – Unique identifier ("origin source" and "version") – Assigned cryptographic algorithm specification – Allowed usage restrictions - Configure [CRYPTO Tester] and [CRYPTOApp01] to share two symmetric keys <SK1> and <SK2>, both with allowed usage flags "kAllowExporting" and "kAllowImporting" enabled for importing/exporting <SecureKM1>. - When testing <SecureKM1>, configure [CRYPTO Tester] to have <SecureKM1Exported1> as exported <SecureKM1> using <SK1>, and <SecureKM1Exported2> as exported <SecureKM1> using <SK2>. - When testing <UnsecureKey1>, configure [CRYPTO Tester] to have <UnsecureKey1Exported> as exported format of <UnsecureKey1>.	
Summary	When testing <SecureKM1>: [CRYPTO Tester] tests whether [CRYPTOApp01] can import <SecureKM1Exported1> using <SK1>. [CRYPTOApp01] imports <SecureKM1Exported> twice, by passing argument "isExportable" of ImportSecureObject API with value "true" to obtain <SecureKM1Exportable>, and by passing "isExportable" with value "false" to obtain <SecureKM1NotExportable>. [CRYPTO Tester] tests whether [CRYPTOApp01] can export <SecureKM1Exportable> using <SK2> including its meta information, and whether [CRYPTOApp01] cannot export <SecureKM1NotExportable>. When testing <UnsecureKey1>: [CRYPTO Tester] tests whether [CRYPTOApp01] can import <UnsecureKey1Exported>. [CRYPTOApp01] imports <UnsecureKey1Exported> to obtain <UnsecureKey1Exportable> (isExportable is not handled in this case because public key material is supposed to be always exportable). [CRYPTO Tester] tests whether [CRYPTOApp01] can export <UnsecureKey1Exportable> including its meta information. - Whether to compare exported key material (<SecureKM1Exportable2> or <UnsecureKey1Exported>) in [CRYPTOApp01] or [CRYPTO Tester] is up to implementer.	
Pre-conditions	- Crypto stack and [CRYPTOApp01] are initialized with key and algorithm. - Communication between [CRYPTO Tester] and [CRYPTOApp01] has been set up.	
Post-conditions	Communication between [CRYPTO Tester] and [CRYPTOApp01] is closed.	
Main Test Execution		
Test Steps		**Pass Criteria**
Step 1	[CRYPTO Tester] Trigger import of key material by sending either <SecureKM1Exported1> or <UnsecureKey1Exported> to [CRYPTOApp01].	
Step 2	[CRYPTOApp01] When testing <SecureKM1>: Import <SecureKM1Exported1> using <SK1> in two ways by: 1. passing argument "isExportable" of ImportSecuredObject API with value "true" to obtain <SecureKM1Exportable>. 2. passing argument "isExportable" of ImportSecuredObject API with value "false" to obtain <SecureKM1NotExportable>. When testing <UnsecureKey1>: Import <UnsecureKey1Exported> to obtain <UnsecureKey1Exportable>.	

▽

△

Step 3	[CRYPTOApp01] Return status (success/failure) of importing key material to [CRYPTO Tester].	
Step 4	[CRYPTO Tester] Check status of importing key material.	[CRYPTO Tester] Status contains success and no error.
Step 5	[CRYPTO Tester] Send trigger of exporting either <SecureKM1Exportable> (using <SK2>), or <UnsecureKey1Exportable> to [CRYPTOApp01].	
Step 6	[CRYPTOApp01] Either export <SecureKM1Exportable> using <SK2> to obtain <SecureKM1Exported2'>, or export <UnsecureKey1Exportable> to obtain <UnsecureKey1Exported'>.	
Step 7	[CRYPTOApp01] Return status (success/failure) of exporting key material to [CRYPTO Tester].	
Step 8	[CRYPTO Tester] Check status of exporting key material.	[CRYPTO Tester] Status contains success and no error.
Step 9	[CRYPTO Tester] Send either <SecureKM1Exported2>, or <UnsecureKey1Exported> to [CRYPTOApp01] for comparison (including meta information).	
Step 10	[CRYPTOApp01] Compare either <SecureKM1Exported2'> with <SecureKM1Exported2>, or <UnsecureKey1Exported'> with <UnsecureKey1Exported>.	
Step 11	[CRYPTOApp01] Return result (matched/unmatched) of key material comparison to [CRYPTO Tester].	
Step 12	[CRYPTO Tester] Check comparison result.	[CRYPTO Tester] Comparison result is "matched."
Step 13	[CRYPTO Tester] Send trigger of exporting <SecureKM1NotExportable> (using <SK2>) to [CRYPTOApp01]. NOTE: This test step and on only applies to <SecureKM1>.	
Step 14	[CRYPTOApp01] Export <SecureKM1NotExportable> using <SK2>.	
Step 15	[CRYPTOApp01] Return status (success/failure) of exporting <SecureKM1NotExportable> to [CRYPTO Tester].	
Step 16	[CRYPTO Tester] Check status of exporting <SecureKM1NotExportable>.	[CRYPTO Tester] Status contains failure or error.

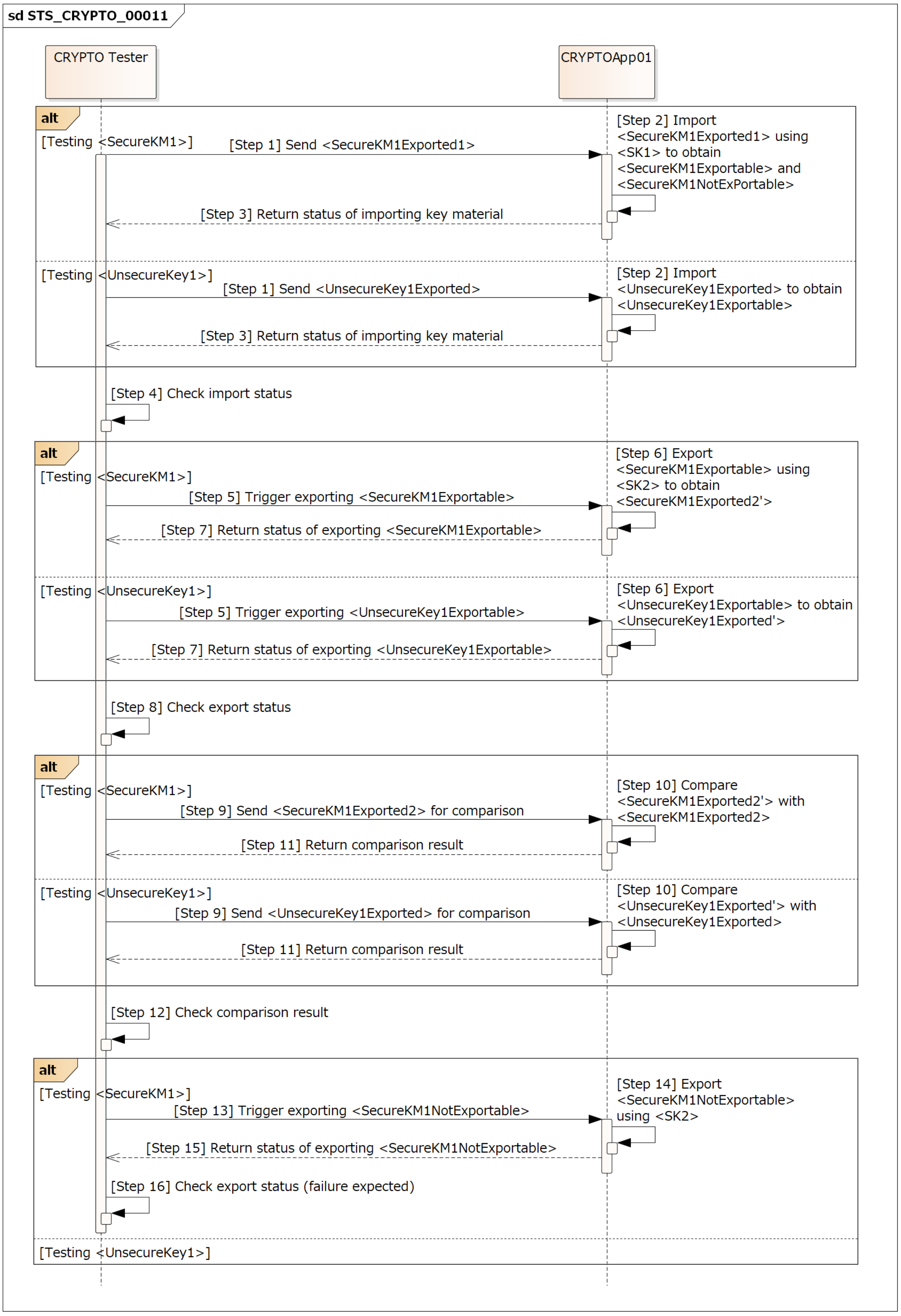
sd STS_CRYPTO_00011
CRYPTO Tester
CRYPTOApp01
alt
[Testing <SecureKM1>]
[Step 1] Send <SecureKM1Exported1>
[Step 2] Import <SecureKM1Exported1> using <SK1> to obtain <SecureKM1Exportable> and <SecureKM1NotExPortable>
[Step 3] Return status of importing key material
[Testing <UnsecureKey1>]
[Step 1] Send <UnsecureKey1Exported>
[Step 2] Import <UnsecureKey1Exported> to obtain <UnsecureKey1Exportable>
[Step 3] Return status of importing key material
[Step 4] Check import status
alt
[Testing <SecureKM1>]
[Step 5] Trigger exporting <SecureKM1Exportable>
[Step 6] Export <SecureKM1Exportable> using <SK2> to obtain <SecureKM1Exported2'>
[Step 7] Return status of exporting <SecureKM1Exportable>
[Testing <UnsecureKey1>]
[Step 5] Trigger exporting <UnsecureKey1Exportable>
[Step 6] Export <UnsecureKey1Exportable> to obtain <UnsecureKey1Exported'>
[Step 7] Return status of exporting <UnsecureKey1Exportable>
[Step 8] Check export status
alt
[Testing <SecureKM1>]
[Step 9] Send <SecureKM1Exported2> for comparison
[Step 10] Compare <SecureKM1Exported2'> with <SecureKM1Exported2>
[Step 11] Return comparison result
[Testing <UnsecureKey1>]
[Step 9] Send <UnsecureKey1Exported> for comparison
[Step 10] Compare <UnsecureKey1Exported'> with <UnsecureKey1Exported>
[Step 11] Return comparison result
[Step 12] Check comparison result
alt
[Testing <SecureKM1>]
[Step 13] Trigger exporting <SecureKM1NotExportable>
[Step 14] Export <SecureKM1NotExportable> using <SK2>
[Step 15] Return status of exporting <SecureKM1NotExportable>
[Step 16] Check export status (failure expected)
[Testing <UnsecureKey1>]

Figure 15.10: Sequence diagram of STS_CRYPTO_00011.

15.2.12 [STS_CRYPTO_00012] Generation/derivation of cryptographic keys and secret seeds.

Test Objective	Verify that Crypto Stack correctly generates cryptographic keys and secret seeds.		
ID	STS_CRYPTO_00012	**State**	Draft
Affected Functional Cluster	Cryptography		
Trace to RS Criteria	[RS_CRYPTO_02101], [RS_CRYPTO_02102], [RS_CRYPTO_02103], [RS_CRYPTO_02003], [RS_CRYPTO_02007], [RS_CRYPTO_02107], [RS_CRYPTO_02111], [RS_CRYPTO_02113], [RS_CRYPTO_02115], [RS_CRYPTO_02309]		
Reference to Test Environment	STC_CRYPTO_00001 in Test configurations		
Configuration Parameters	- Configure [CRYPTO Tester] to have information necessary to generate or derive a key material <KM1Exportable> (i.e. key or secret seed): – used algorithm <AlgId1> – allowed usage <Usage1> (e.g. "kAllowDataEncryption") – whether generating/deriving a session key/secret seed <IsSession1> – whether generating/deriving an exportable key/secret seed <IsExportable1> (see note below) – source key material <SrcKM1> (when testing derivation of key/secret seed) – salt <Salt1> (when testing derivation of key/secret seed) – number of iterations <Iteration1> (when testing derivation of key/secret seed) - Configure [CRYPTO Tester] and [CRYPTOApp01] to share a symmetric key <SK1> with allowed usage flag "kAllowExporting" enabled for exporting <KM1Exportable>. NOTE: <IsExportable1> must be configured "true" to pass all test steps in this test case. Configuring <IsExportable1> to "false" can test whether Crypto Stack generates nonexportable keys/secret seeds, in which case Step 8 must fail and Step 9 and all further test steps would be invalid.		
Summary	[CRYPTO Tester] checks whether [CRYPTOApp01] correctly generates/derivates an exportable key material <KM1Exportable> (i.e. key or secret seed), and checks whether generated/derivated <KM1Exportable> can be used correctly by exporting <KM1Exportable> from [CRYPTOApp01] to [CRYPTO Tester], executing allowed usage <Usage1> of <KM1Exportable> on both sides, and comparing the results <Result1> and <Result1'>. - Whether to compare <Result1> in [CRYPTOApp01] or [CRYPTO Tester] is up to implementer.		
Pre-conditions	Communication between [CRYPTO Tester] and [CRYPTOApp01] has been set up.		
Post-conditions	Communication between [CRYPTO Tester] and [CRYPTOApp01] is closed.		
Main Test Execution			
Test Steps		**Pass Criteria**	
Step 1	[CRYPTO Tester] Trigger generation/derivation of <KM1Exportable> by sending <AlgId1>, <Usage1>, <IsSession1>, <IsExportable1>, <SrcKM1>, <Salt1>, and <Iteration1> to [CRYPTOApp01].		
Step 2	[CRYPTOApp01] Generate/derive <KM1Exportable> using <AlgId1>, <Usage1>, <IsSession1>, <IsExportable1>, <SrcKM1>, <Salt1>, and <Iteration1>. NOTE: Exportable key/secret seed is generated by passing the argument "isExportable" with value "true" to Crypto API GenerateSymmetricKey/GeneratePrivatekey/DeriveKey/GenerateSeed/DeriveSeed.		

▽

△

Step 3	[CRYPTOApp01] Return <KM1Exportable> generation/derivation status (success/failure) to [CRYPTO Tester].	
Step 4	[CRYPTO Tester] Check <KM1Exportable> generation/derivation status.	[CRYPTO Tester] Status contains success and no error.
Step 5	[CRYPTO Tester] Send trigger of exporting <KM1Exportable> to [CRYPTOApp01].	
Step 6	[CRYPTOApp01] Export <KM1Exportable> (using <SK1>) to obtain <KM1Exported> (i.e. <KM1Exportable> in an exported format).	
Step 7	[CRYPTOApp01] Return <KM1Exportable> export status (success/failure) and <KM1Exported> to [CRYPTO Tester].	
Step 8	[CRYPTO Tester] Check <KM1Exportable> export status.	[CRYPTO Tester] Status contains success and no error.
Step 9	[CRYPTO Tester] Import <KM1Exported> to obtain <KM1Imported>.	[CRYPTO Tester] <KM1Exported> is imported with success and no error.
Step 10	[CRYPTO Tester] Trigger [CRYPTOApp01] to execute <Usage1> of <KM1Exportable>.	
Step 11	[CRYPTOApp01] Execute <Usage1> using <KM1Exportable> to obtain <Result1'>.	
Step 12	[CRYPTOApp01] Return execution status to [CRYPTO Tester].	
Step 13	[CRYPTO Tester] Check execution status.	[CRYPTO Tester] Status contains success and no error.
Step 14	[CRYPTO Tester] Execute <Usage1> using <KM1Imported> in the same way as [CRYPTOApp01] to obtain <Result1>.	[CRYPTO Tester] Execution status contains success and no error.
Step 15	[CRYPTO Tester] Send <Result1> to [CRYPTOApp01] for comparison.	
Step 16	[CRYPTOApp01] Compare <Result1'> with <Result1>.	
Step 17	[CRYPTOApp01] Return comparison result (matched/unmatched) to [CRYPTO Tester].	
Step 18	[CRYPTO Tester] Check comparison result.	[CRYPTO Tester] Comparison result is "matched."

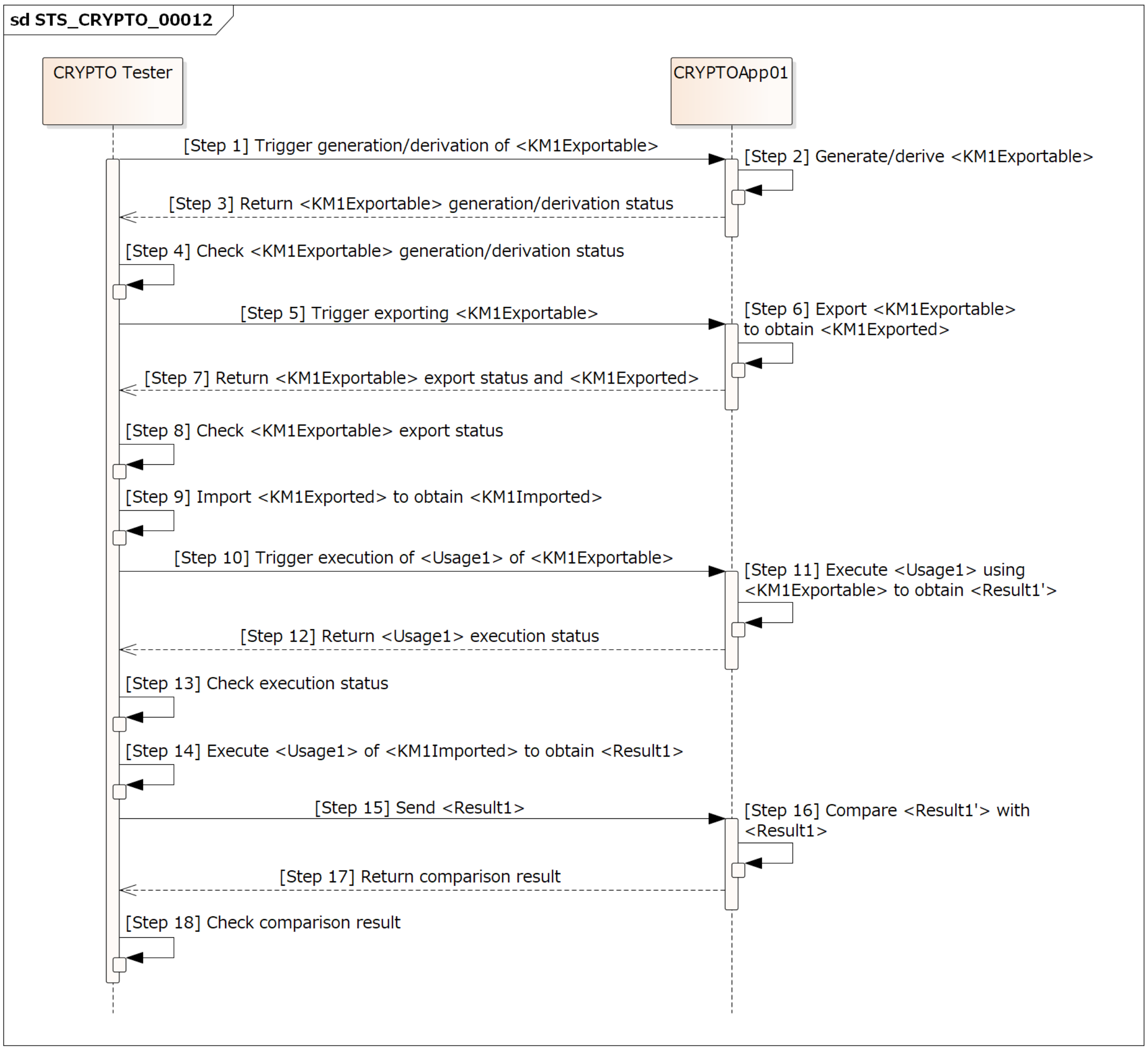
sd STS_CRYPTO_00012
CRYPTO Tester
CRYPTOApp01
[Step 1] Trigger generation/derivation of <KM1Exportable>
[Step 2] Generate/derive <KM1Exportable>
[Step 3] Return <KM1Exportable> generation/derivation status
[Step 4] Check <KM1Exportable> generation/derivation status
[Step 5] Trigger exporting <KM1Exportable>
[Step 6] Export <KM1Exportable> to obtain <KM1Exported>
[Step 7] Return <KM1Exportable> export status and <KM1Exported>
[Step 8] Check <KM1Exportable> export status
[Step 9] Import <KM1Exported> to obtain <KM1Imported>
[Step 10] Trigger execution of <Usage1> of <KM1Exportable>
[Step 11] Execute <Usage1> using <KM1Exportable> to obtain <Result1'>
[Step 12] Return <Usage1> execution status
[Step 13] Check execution status
[Step 14] Execute <Usage1> of <KM1Imported> to obtain <Result1>
[Step 15] Send <Result1>
[Step 16] Compare <Result1'> with <Result1>
[Step 17] Return comparison result
[Step 18] Check comparison result

Figure 15.11: Sequence diagram of STS_CRYPTO_00012.

15.2.13 [STS_CRYPTO_00013] PKI/X.509 - handling of certificate signing request (CSR) and certificates.

Test Objective	Verify that Crypto Stack correctly handles certificate signing request (CSR) and certificates.		
ID	STS_CRYPTO_00013	**State**	Draft
Affected Functional Cluster	Cryptography		
Trace to RS Criteria	[RS_CRYPTO_02306], [RS_CRYPTO_02115]		
Reference to Test Environment	STC_CRYPTO_00001		

▽

△

Configuration Parameters	- Configure [CRYPTO Tester] to have an asymmetric key pair: public key <TPbK1> and private key <TPvK1>, for creation of an end-entity certificate <EECERT1>. - Configure [CRYPTO Tester] to have an intermediate certificate <IMCERT1>. - Configure [CRYPTO Tester] to have an expected certificate signing request <CSR1> to be compared with <CSR1'> created by [CRYPTOApp01]. - Configure [CRYPTOApp01] to have a root certificate <RCERT1> installed in certificate-slot and accessible as a "root of trust". - Configure [CRYPTOApp01] to have an asymmetric key pair: a public key <APbK1> and a private key <APvK1>, and distinguished name <DN1> for creation of certificate signing request <CSR1'>. - Configure [CRYPTOApp01] to have "CA Connector" permissions.	
Summary	[CRYPTO Tester] checks whether [CRYPTOApp01] correctly: 1. creates and exports certificate signing request <CSR1>. 2. verifies <RCERT1>-<IMCERT1>-<EECERT1> certificate chain. 3. imports, exports, and removes <EECERT1>. Verification of certificate chain is first done with missing <IMCERT1> in [CRYPTOApp01] expecting failure, and then with valid <IMCERT1> expecting success.	
Pre-conditions	Communication between [CRYPTO Tester] and [CRYPTOApp01] has been set up.	
Post-conditions	Communication between [CRYPTO Tester] and [CRYPTOApp01] is closed.	
Main Test Execution		
Test Steps		**Pass Criteria**
Step 1	[CRYPTO Tester] Trigger creating certificate signing request <CSR1'>.	
Step 2	[CRYPTOApp01] Create <CSR1'> using <DN1>, <APbK1>, and <APvK1>.	
Step 3	[CRYPTOApp01] Return status (success/failure) of creating <CSR1'> to [CRYPTO Tester].	
Step 4	[CRYPTO Tester] Check status of creating <CSR1'>.	[CRYPTO Tester] Status contains success and no error.
Step 5	[CRYPTO Tester] Trigger exporting <CSR1'>.	
Step 6	[CRYPTOApp01] Export <CSR1'>.	
Step 7	[CRYPTOApp01] Return export status (success/failure) and exported <CSR1'> to [CRYPTO Tester].	
Step 8	[CRYPTO Tester] Check status of exporting <CSR1'>.	[CRYPTO Tester] Status has success and no error.
Step 9	[CRYPTO Tester] Check <CSR1'> by comparing <CSR1'> with <CSR1>.	[CRYPTO Tester] <CSR1'> matches <CSR1>.
Step 10	[CRYPTO Tester] Trigger setting "Pending" status to <CSR1'>.	
Step 11	[CRYPTOApp01] Set "Pending" status to <CSR1'>.	

▽

△

Step 12	[CRYPTOApp01] Return status (success/failure) of setting "Pending" status to [CRYPTOTester].	
Step 13	[CRYPTO Tester] Check status of setting "Pending" status.	[CRYPTO Tester] Status contains success and no error.
Step 14	[CRYPTO Tester] Trigger parsing <EECERT1> by sending <EECERT1> to [CRYPTOApp01].	
Step 15	[CRYPTOApp01] Parse <EECERT1>.	
Step 16	[CRYPTOApp01] Return status (success/failure) of parsing <EECERT1> to [CRYPTO Tester].	
Step 17	[CRYPTO Tester] Check status of parsing <EECERT1>.	[CRYPTO Tester] Status contains success and no error.
Step 18	[CRYPTO Tester] Send trigger of verifying <RCERT1>-<EECERT1> chain to [CRYPTOApp01].	
Step 19	[CRYPTOApp01] Verify <RCERT1>-<EECERT1> certificate chain.	
Step 20	[CRYPTOApp01] Retrieve statuses of <RCERT1> and <EECERT1> using Certificate::GetStatus API.	
Step 21	[CRYPTOApp01] Return verification status of <RCERT1>-<EECERT1> chain and statuses of <RCERT1> and <EECERT1> to [CRYPTO Tester].	
Step 22	[CRYPTO Tester] Check verification status of <RCERT1>-<EECERT1> chain and statuses of <RCERT1> and <EECERT1>.	[CRYPTO Tester] Verification status of <RCERT1>-<EECERT1> chain is "kNoTrust", statuses of <RCERT1> and <EECERT1> are "kValid" and "kNoTrust", respectively. NOTE: The API VerifyCertChain updates status of <EECERT1> to "kNoTrust" because <IMCERT1> referenced by <EECERT1> is missing on the [CRYPTOApp01] side.
Step 23	[CRYPTO Tester] Trigger importing <EECERT1> to (non-)volatile storage.	
Step 24	[CRYPTOApp01] Import <EECERT1> to (non-)volatile storage.	
Step 25	[CRYPTOApp01] Return status (success/failure) of importing <EECERT1> to [CRYPTO Tester].	
Step 26	[CRYPTO Tester] Check status of importing <EECERT1>.	[CRYPTO Tester] Status contains success and no error.

▽

△

Step 27	[CRYPTO Tester] Trigger parsing <IMCERT1> by sending <IMCERT1> to [CRYPTOApp01].	
Step 28	[CRYPTOApp01] Parse <IMCERT1>.	
Step 29	[CRYPTOApp01] Return status (success/failure) of parsing <IMCERT1> to [CRYPTO Tester].	
Step 30	[CRYPTO Tester] Check status of parsing <IMCERT1>.	[CRYPTO Tester] Status contains success and no error.
Step 31	[CRYPTO Tester] Send trigger of verifying <RCERT1>-<IMCERT1>-<EECERT1> certificate chain to [CRYPTOApp01].	
Step 32	[CRYPTOApp01] Verify <RCERT1>-<IMCERT1>-<EECERT1> certificate chain.	
Step 33	[CRYPTOApp01] Return verification status of <RCERT1>-<IMCERT1>-<EECERT1> certificate chain to [CRYPTO Tester].	
Step 34	[CRYPTO Tester] Check verification status of certificate chain.	[CRYPTO Tester] Verification status of certificate chain is "kValid."
Step 35	[CRYPTO Tester] Trigger loading and exporting <EECERT1> from (non-)volatile storage.	
Step 36	[CRYPTOApp01] Load and export <EECERT1> from (non-)volatile storage.	
Step 37	[CRYPTOApp01] Return <EECERT1> to [CRYPTO Tester].	
Step 38	[CRYPTO Tester] Verify <EECERT1> using <TPbK1> retrieved from <EECERT1>.	[CRYPTO Tester] <EECERT1> is valid.
Step 39	[CRYPTO Tester] Send trigger of removing <EECERT1> to [CRYPTOApp01].	
Step 40	[CRYPTOApp01] Remove <EECERT1> from (non-)volatile storage.	
Step 41	[CRYPTOApp01] Return status (success/failure) of removing <EECERT1> to [CRYPTO Tester].	
Step 42	[CRYPTO Tester] Check status of removing <EECERT1>.	[CRYPTO Tester] Status contains success and no error.
Step 43	[CRYPTO Tester] Trigger loading <EECERT1> from (non-)volatile storage .	

▽

△

Step 44	[CRYPTOApp01] Load <EECERT1> from (non-)volatile storage.	
Step 45	[CRYPTOApp01] Return pointer value of <EECERT1> to [CRYPTO Tester].	
Step 46	[CRYPTO Tester] Check pointer value of <EECERT1>.	[CRYPTO Tester] Pointer value of <EECERT1> is nullptr.

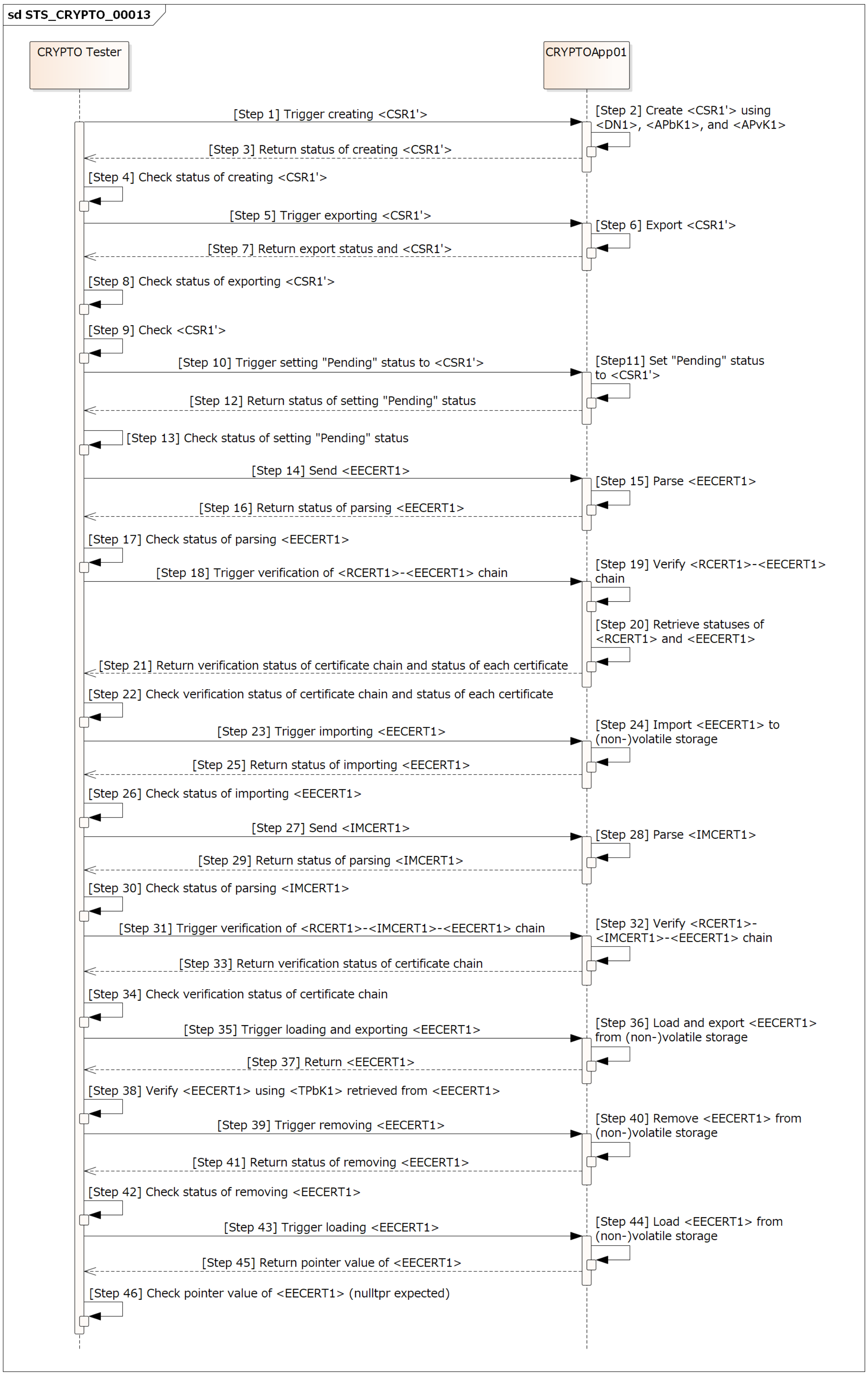
sd STS_CRYPTO_00013
CRYPTO Tester
CRYPTOApp01
[Step 1] Trigger creating <CSR1'>
[Step 2] Create <CSR1'> using <DN1>, <APbK1>, and <APvK1>
[Step 3] Return status of creating <CSR1'>
[Step 4] Check status of creating <CSR1'>
[Step 5] Trigger exporting <CSR1'>
[Step 6] Export <CSR1'>
[Step 7] Return export status and <CSR1'>
[Step 8] Check status of exporting <CSR1'>
[Step 9] Check <CSR1'>
[Step 10] Trigger setting "Pending" status to <CSR1'>
[Step11] Set "Pending" status to <CSR1'>
[Step 12] Return status of setting "Pending" status
[Step 13] Check status of setting "Pending" status
[Step 14] Send <EECERT1>
[Step 15] Parse <EECERT1>
[Step 16] Return status of parsing <EECERT1>
[Step 17] Check status of parsing <EECERT1>
[Step 18] Trigger verification of <RCERT1>-<EECERT1> chain
[Step 19] Verify <RCERT1>-<EECERT1> chain
[Step 20] Retrieve statuses of <RCERT1> and <EECERT1>
[Step 21] Return verification status of certificate chain and status of each certificate
[Step 22] Check verification status of certificate chain and status of each certificate
[Step 23] Trigger importing <EECERT1>
[Step 24] Import <EECERT1> to (non-)volatile storage
[Step 25] Return status of importing <EECERT1>
[Step 26] Check status of importing <EECERT1>
[Step 27] Send <IMCERT1>
[Step 28] Parse <IMCERT1>
[Step 29] Return status of parsing <IMCERT1>
[Step 30] Check status of parsing <IMCERT1>
[Step 31] Trigger verification of <RCERT1>-<IMCERT1>-<EECERT1> chain
[Step 32] Verify <RCERT1>-<IMCERT1>-<EECERT1> chain
[Step 33] Return verification status of certificate chain
[Step 34] Check verification status of certificate chain
[Step 35] Trigger loading and exporting <EECERT1>
[Step 36] Load and export <EECERT1> from (non-)volatile storage
[Step 37] Return <EECERT1>
[Step 38] Verify <EECERT1> using <TPbK1> retrieved from <EECERT1>
[Step 39] Trigger removing <EECERT1>
[Step 40] Remove <EECERT1> from (non-)volatile storage
[Step 41] Return status of removing <EECERT1>
[Step 42] Check status of removing <EECERT1>
[Step 43] Trigger loading <EECERT1>
[Step 44] Load <EECERT1> from (non-)volatile storage
[Step 45] Return pointer value of <EECERT1>
[Step 46] Check pointer value of <EECERT1> (nulltpr expected)

Figure 15.12: Sequence diagram of STS_CRYPTO_00013.

15.2.14 [STS_CRYPTO_00014] PKI/X.509 - verification of certificates with certificate revocation list (CRL) and by online certificate status protocol (OCSP).

Test Objective	Verify that Crypto Stack correctly verifies and updates state of certificates according to certificate revocation list (CRL) and online certificate status protocol (OCSP).		
ID	STS_CRYPTO_00014	**State**	Draft
Affected Functional Cluster	Cryptography		
Trace to RS Criteria	[RS_CRYPTO_02306], [RS_CRYPTO_02115]		
Reference to Test Environment	STC_CRYPTO_00001		
Configuration Parameters	- Configure [CRYPTO Tester] to have three asymmetric key pairs: 1. public key <TPbK1> and private key <TPvK1> for creation of signed OCSP responses <OCSPResp1> and <OCSPResp2>. 2. public key <TPbK2> and private key <TPvK2> for creation of a root certificate <RCERT2> which contain same information as <RCERT1> below but with different asymmetric key pair. 3. public key <TPbK3> and private key <TPvK3> for creation of an end-entity certificate <EECERT2> which contain same information as <EECERT1> below but with different asymmetric key pair. - Configure [CRYPTO Tester] to have certificate revocation list <CRL1> containing revocation of <RCERT1>. - Configure [CRYPTO Tester] to have expected <OCSPReq1> and <OCSPReq2> to be compared with <OCSPReq1'> and <OCSPReq2'> created by [CRYPTOApp01]. - Configure [CRYPTOApp01] to have a root certificate <RCERT1> installed in certificate-slot and accessible as "root of trust". - Configure [CRYPTOApp01] to have an intermediate certificate <IMCERT1> and an end-entity certificate <EECERT1> installed in certificate-slot and accessible. - Configure [CRYPTOApp01] to have "Trust Master" permission.		
Summary	[CRYPTO Tester] checks whether [CRYPTOApp01] correctly: 1. imports certificate revocation list <CRL1>. 2. detects invalid certificate chain with revoked <RCERT1> by <CRL1> and with revoked <EECERT1> by OCSP. 3. imports a valid root certificate <RCERT2> and applies "set as root of trust." 4. imports a valid end-entity certificate <EECERT2>." 5. verifies <RCERT2>-<IMCERT1>-<EECERT2> certificate chain with <CRL1> and by OCSP.		
Pre-conditions	[CRYPTO Tester] has <RCERT2> created using distinguished name <DN1>, public key <TPbK2>, and private key <TPvK2>. [CRYPTO Tester] has <EECERT2> created using distinguished name <DN2>, public key <TPbK3>, and private key <TPvK3>. Communication between [CRYPTO Tester] and [CRYPTOApp01] has been set up.		
Post-conditions	Communication between [CRYPTO Tester] and [CRYPTOApp01] is closed.		
Main Test Execution			
Test Steps		**Pass Criteria**	
Step 1	[CRYPTO Tester] Send <CRL1>, containing revocation of <RCERT1>, to [CRYPTOApp01].		
Step 2	[CRYPTOApp01] Import <CRL1>.		
Step 3	[CRYPTOApp01] Return status (success/failure) of importing <CRL1> to [CRYPTO Tester].		

▽

△

Step 4	[CRYPTO Tester] Check status of importing <CRL1>.	[CRYPTO Tester] Status contains success and no error.
Step 5	[CRYPTO Tester] Trigger verifying <RCERT1>-<IMCERT1>-<EECERT1> certificate chain.	
Step 6	[CRYPTOApp01] Verify <RCERT1>-<IMCERT1>-<EECERT1> certificate chain.	
Step 7	[CRYPTOApp01] Retrieve statuses of <RCERT1>, <IMCERT1>, and <EECERT1>.	
Step 8	[CRYPTOApp01] Return verification status of <RCERT1>-<IMCERT1>-<EECERT1> certificate chain and statuses of <RCERT1>, <IMCERT1>, and <EECERT1> to [CRYPTO Tester].	
Step 9	[CRYPTO Tester] Check verification status of <RCERT1>-<IMCERT1>-<EECERT1> certificate chain and statuses of <RCERT1>, <IMCERT1>, and <EECERT1>.	[CRYPTO Tester] Verification status of <RCERT1>-<IMCERT1>-<EECERT1> certificate chain is "kInvalid", statuses of <RCERT1>, <IMCERT1>, and <EECERT1> are "kInvalid", "kNoTrust", and "kNoTrust", respectively.
Step 10	[CRYPTO Tester] Trigger creating <OCSPReq1'>.	
Step 11	[CRYPTOApp01] Create <OCSPReq1'> using <RCERT1>, <IMCERT1>, and <EECERT1>.	
Step 12	[CRYPTOApp01] Return status of creating <OCSPReq1'> to [CRYPTO Tester].	
Step 13	[CRYPTO Tester] Check status of creating <OCSPReq1'>.	[CRYPTO Tester] Status contains success and no error.
Step 14	[CRYPTO Tester] Trigger exporting <OCSPReq1'>.	
Step 15	[CRYPTOApp01] Export <OCSPReq1'>.	
Step 16	[CRYPTOApp01] Return <OCSPReq1'> to [CRYPTO Tester].	
Step 17	[CRYPTO Tester] Check <OCSPReq1'> by comparing <OCSPReq1'> with <OCSPReq1>.	[CRYPTO Tester] <OCSPReq1'> matches <OCSPReq1>.
Step 18	[CRYPTO Tester] Trigger retrieving statuses of <RCERT1>, <IMCERT1>, and <EECERT1> by sending <OCSPResp1>, containing revocation of <RCERT1> and <EECERT1>, to [CRYPTOApp01].	

▽

△

Step 19	[CRYPTOApp01] Retrieve verification statuses of <RCERT1>, <IMCERT1>, and <EECERT1> from <OCSPResp1>.	
Step 20	[CRYPTOApp01] Return statuses of <RCERT1>, <IMCERT1>, and <EECERT1> to [CRYPTO Tester].	
Step 21	[CRYPTO Tester] Check statuses of <RCERT1>, <IMCERT1>, and <EECERT1>.	[CRYPTO Tester] Statuses of <RCERT1>, <IMCERT1>, and <EECERT1> are "kInvalid", "kNoTrust", and "kInvalid", respectively.
Step 22	[CRYPTO Tester] Trigger importing <RCERT2> by sending <RCERT2> to [CRYPTOApp01].	
Step 23	[CRYPTOApp01] Import <RCERT2> to non-volatile storage.	
Step 24	[CRYPTOApp01] Return status (success/failure) of importing <RCERT2> to [CRYPTO Tester].	
Step 25	[CRYPTO Tester] Check status of importing <RCERT2>.	[CRYPTO Tester] Status contains success and no error.
Step 26	[CRYPTO Tester] Trigger applying "set as root of trust" to <RCERT2>.	
Step 27	[CRYPTOApp01] Apply "set as root of trust" to <RCERT2>.	
Step 28	[CRYPTOApp01] Return status (success/failure) of applying "set as root of trust" to [CRYPTO Tester].	
Step 29	[CRYPTO Tester] Check status of applying "set as root of trust".	[CRYPTO Tester] Status contains success and no error.
Step 30	[CRYPTO Tester] Trigger importing <EECERT2> by sending <EECERT2> to [CRYPTOApp01].	
Step 31	[CRYPTOApp01] Import <EECERT2> to (non-)volatile storage.	
Step 32	[CRYPTOApp01] Return status (success/failure) of importing <EECERT2> to [CRYPTO Tester].	
Step 33	[CRYPTO Tester] Check status of importing <EECERT2>.	[CRYPTO Tester] Status contains success and no error.
Step 34	[CRYPTO Tester] Trigger verifying <RCERT2>-<IMCERT1>-<EECERT2> certificate chain.	
Step 35	[CRYPTOApp01] Verify <RCERT2>-<IMCERT1>-<EECERT2> certificate chain.	

▽

△

Step 36	[CRYPTOApp01] Retrieve statuses of <RCERT2>, <IMCERT1>, and <EECERT2>.	
Step 37	[CRYPTOApp01] Return verification status of <RCERT2>-<IMCERT1>-<EECERT2> certificate chain and statuses of <RCERT2>, <IMCERT1>, and <EECERT2> to [CRYPTO Tester].	
Step 38	[CRYPTO Tester] Check verification status of <RCERT2>-<IMCERT1>-<EECERT2> chain and statuses of <RCERT2>, <IMCERT1>, and <EECERT2>.	[CRYPTO Tester] Verification status of <RCERT2>-<IMCERT1>-<EECERT2> certificate chain is "kValid", statuses of <RCERT2>, <IMCERT1>, and <EECERT2> are "kValid", "kValid", and "kValid", respectively.
Step 39	[CRYPTO Tester] Trigger creating <OCSPReq2'>.	
Step 40	[CRYPTOApp01] Create <OCSPReq2'> using <RCERT2>, <IMCERT1>, and <EECERT2>.	
Step 41	[CRYPTOApp01] Return status of creating <OCSPReq2'> to [CRYPTO Tester].	
Step 42	[CRYPTO Tester] Check status of creating <OCSPReq2'>.	[CRYPTO Tester] Status contains success and no error.
Step 43	[CRYPTO Tester] Trigger exporting <OCSPReq2'>.	
Step 44	[CRYPTOApp01] Export <OCSPReq2'>.	
Step 45	[CRYPTOApp01] Return <OCSPReq2'> to [CRYPTO Tester].	
Step 46	[CRYPTO Tester] Check <OCSPReq2'> by comparing <OCSPReq2'> with <OCSPReq2>.	[CRYPTO Tester] <OCSPReq2'> matches <OCSPReq2>.
Step 47	[CRYPTO Tester] Trigger retrieving statuses of <RCERT2>, <IMCERT1>, and <EECERT2> by sending <OCSPResp2>, containing revocation of <IMCERT1>, to [CRYPTOApp01].	
Step 48	[CRYPTOApp01] Retrieve verification statuses of <RCERT2>, <IMCERT1>, and <EECERT2> from <OCSPResp2>.	
Step 49	[CRYPTOApp01] Return statuses of <RCERT2>, <IMCERT1>, and <EECERT2> to [CRYPTO Tester].	
Step 50	[CRYPTO Tester] Check statuses of <RCERT2>, <IMCERT1>, and <EECERT2>.	[CRYPTO Tester] Statuses of <RCERT2>, <IMCERT1>, and <EECERT2> are "kValid", "kInvalid", and "kNoTrust", respectively.

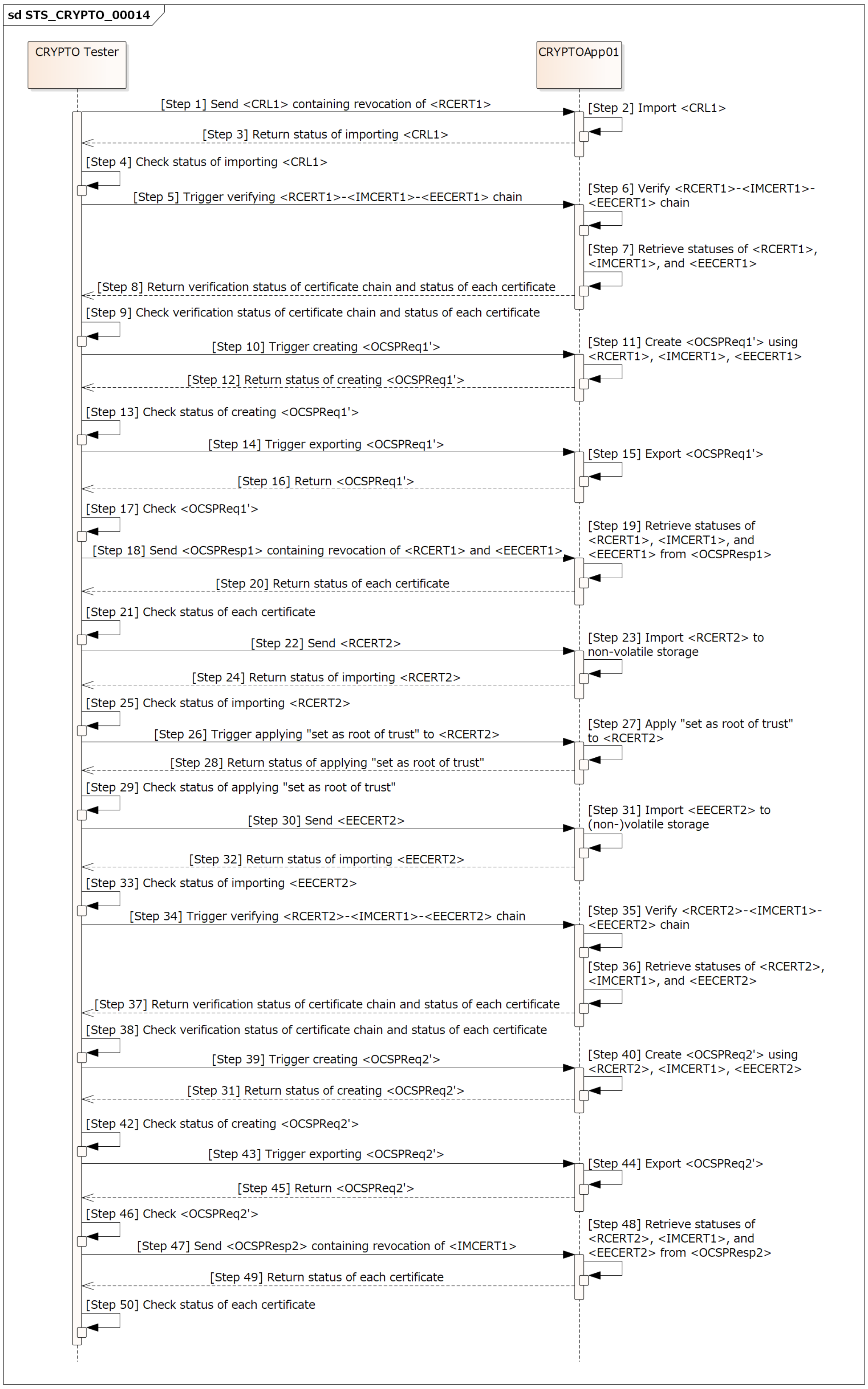
sd STS_CRYPTO_00014
CRYPTO Tester
CRYPTOApp01
[Step 1] Send <CRL1> containing revocation of <RCERT1>
[Step 2] Import <CRL1>
[Step 3] Return status of importing <CRL1>
[Step 4] Check status of importing <CRL1>
[Step 5] Trigger verifying <RCERT1>-<IMCERT1>-<EECERT1> chain
[Step 6] Verify <RCERT1>-<IMCERT1>-<EECERT1> chain
[Step 7] Retrieve statuses of <RCERT1>, <IMCERT1>, and <EECERT1>
[Step 8] Return verification status of certificate chain and status of each certificate
[Step 9] Check verification status of certificate chain and status of each certificate
[Step 10] Trigger creating <OCSPReq1'>
[Step 11] Create <OCSPReq1'> using <RCERT1>, <IMCERT1>, <EECERT1>
[Step 12] Return status of creating <OCSPReq1'>
[Step 13] Check status of creating <OCSPReq1'>
[Step 14] Trigger exporting <OCSPReq1'>
[Step 15] Export <OCSPReq1'>
[Step 16] Return <OCSPReq1'>
[Step 17] Check <OCSPReq1'>
[Step 18] Send <OCSPResp1> containing revocation of <RCERT1> and <EECERT1>
[Step 19] Retrieve statuses of <RCERT1>, <IMCERT1>, and <EECERT1> from <OCSPResp1>
[Step 20] Return status of each certificate
[Step 21] Check status of each certificate
[Step 22] Send <RCERT2>
[Step 23] Import <RCERT2> to non-volatile storage
[Step 24] Return status of importing <RCERT2>
[Step 25] Check status of importing <RCERT2>
[Step 26] Trigger applying "set as root of trust" to <RCERT2>
[Step 27] Apply "set as root of trust" to <RCERT2>
[Step 28] Return status of applying "set as root of trust"
[Step 29] Check status of applying "set as root of trust"
[Step 30] Send <EECERT2>
[Step 31] Import <EECERT2> to (non-)volatile storage
[Step 32] Return status of importing <EECERT2>
[Step 33] Check status of importing <EECERT2>
[Step 34] Trigger verifying <RCERT2>-<IMCERT1>-<EECERT2> chain
[Step 35] Verify <RCERT2>-<IMCERT1>-<EECERT2> chain
[Step 36] Retrieve statuses of <RCERT2>, <IMCERT1>, and <EECERT2>
[Step 37] Return verification status of certificate chain and status of each certificate
[Step 38] Check verification status of certificate chain and status of each certificate
[Step 39] Trigger creating <OCSPReq2'>
[Step 40] Create <OCSPReq2'> using <RCERT2>, <IMCERT1>, <EECERT2>
[Step 31] Return status of creating <OCSPReq2'>
[Step 42] Check status of creating <OCSPReq2'>
[Step 43] Trigger exporting <OCSPReq2'>
[Step 44] Export <OCSPReq2'>
[Step 45] Return <OCSPReq2'>
[Step 46] Check <OCSPReq2'>
[Step 47] Send <OCSPResp2> containing revocation of <IMCERT1>
[Step 48] Retrieve statuses of <RCERT2>, <IMCERT1>, and <EECERT2> from <OCSPResp2>
[Step 49] Return status of each certificate
[Step 50] Check status of each certificate

Figure 15.13: Sequence diagram of STS_CRYPTO_00014.

15.2.15 [STS_CRYPTO_00015] Encryption and decryption of randomly accessed data using "counter mode" stream cipher.

Test Objective	Verify that Crypto Stack correctly encrypts and decrypts randomly accessed data using "counter mode" stream cipher.		
ID	STS_CRYPTO_00015	**State**	Draft
Affected Functional Cluster	Cryptography		
Trace to RS Criteria	[RS_CRYPTO_02304], [RS_CRYPTO_02001], [RS_CRYPTO_02008], [RS_CRYPTO_02115], [RS_CRYPTO_02201], [RS_CRYPTO_02302]		
Reference to Test Environment	STC_CRYPTO_00001 in Test configurations		
Configuration Parameters	- Configure [CRYPTO Tester] and [CRYPTOApp01] to have a common symmetric key <SK1> for symmetric "counter mode" encryption/decryption. - Configure [CRYPTO Tester] to have plaintext data <Plaintext1> and <Plaintext2> which are larger than two encryption blocks. - Configure [CRYPTO Tester] to have ciphertext data <Ciphertext1> and <Ciphertext2> which are larger than two decryption blocks.		
Summary	[CRYPTO Tester] sends <Plaintext1> and <Offset1> to [CRYPTOApp01], and [CRYPTOApp01] encrypts one encryption block in <Plaintext1> starting from <Offset1> using symmetric key <SK1>. [CRYPTO Tester] sends <Offset2> to [CRYPTOApp01], and [CRYPTOApp01] advances encryption postion in <Plaintext1> and state of stream cipher context by <Offset2>. [CRYPTO Tester] triggers the rest of the encryption, and [CRYPTOApp01] continues encryption up to the end of <Plaintext1> to obtain <Ciphertext1'>. <Ciphertext1'> is compared with <Ciphertext1> which is generated in the same way on the [CRYPTO Tester] side. Decryption is tested in similar way as above described encryption. - Value of <Offset1> and <Offset2> are signed integers, multiple of encryption/decryption block size, and chosen within range of <Plaintext1>/<Ciphertext1>. - Data encryption/decryption on the [CRYPTO Tester] side is performed either prior to running test or during a test step. - Whether to compare encryption/decryption result (<Ciphertext1> and <Plaintext2>) in [CRYPTOApp01] or [CRYPTO Tester] is up to implementer.		
Pre-conditions	- Crypto stack and [CRYPTOApp01] are initialized with used key (<SK1>), algorithm, and domain parameter as applicable. - Communication between [CRYPTO Tester] and [CRYPTOApp01] has been set up. - Symmetric key <SK1> can be accessed by [CRYPTOApp01].		
Post-conditions	Communication between [CRYPTO Tester] and [CRYPTOApp01] is closed.		
Main Test Execution			
Test Steps		**Pass Criteria**	
Step 1	[CRYPTO Tester] Trigger encryption of <Plaintext1> by sending <Plaintext1> and <Offset1> to [CRYPTOApp01].		
Step 2	[CRYPTOApp01] Encrypt one encryption block in <Plaintext1> starting from <Offset1> using <SK1>.		
Step 3	[CRYPTOApp01] Return encryption status (success/failure) to [CRYPTO Tester].		
Step 4	[CRYPTO Tester] Check encryption status.	[CRYPTO Tester] Encryption status contains success and no error.	

▽

△

Step 5	[CRYPTO Tester] Trigger "seeking" encryption position of <Plaintext1> by sending <Offset2> to [CRYPTOApp01].	
Step 6	[CRYPTOApp01] Seek encryption position of <Plaintext1> and state of stream cipher context by <Offset2>.	
Step 7	[CRYPTOApp01] Return seek status (success/failure) to [CRYPTO Tester].	
Step 8	[CRYPTO Tester] Check seek status.	[CRYPTO Tester] Seek status contains success and no error.
Step 9	[CRYPTO Tester] Trigger remaining encryption of <Plaintext1>.	
Step 10	[CRYPTOApp01] Resume and complete encryption of <Plaintext1> to obtain <Ciphertext1'>.	
Step 11	[CRYPTOApp01] Return encryption status (success/failure) to [CRYPTO Tester].	
Step 12	[CRYPTO Tester] Check encryption status.	[CRYPTO Tester] Encryption status contains success and no error.
Step 13	[CRYPTO Tester] Send <Ciphertext1> (created in the same way as <Ciphertext1'> on the [CRYPTO Tester] side) to [CRYPTOApp01] for comparison.	
Step 14	[CRYPTOApp01] Compare <Ciphertext1'> with <Ciphertext1>.	
Step 15	[CRYPTOApp01] Return comparison result (matched/unmatched) to [CRYPTO Tester].	
Step 16	[CRYPTO Tester] Check comparison result.	[CRYPTO Tester] Comparison result is "matched."
Step 17	[CRYPTO Tester] Trigger decryption of <Ciphertext2> by sending <Ciphertext2> and <Offset1> to [CRYPTOApp01].	
Step 18	[CRYPTOApp01] Decrypt one decryption block in <Ciphertext2> starting from <Offset1> using <SK1>.	
Step 19	[CRYPTOApp01] Return decryption status (success/failure) to [CRYPTO Tester].	
Step 20	[CRYPTO Tester] Check decryption status.	[CRYPTO Tester] Decryption status contains success and no error.

▽

△

Step 21	[CRYPTO Tester] Trigger "seeking" decryption position of <Ciphertext2> by sending <Offset2> to [CRYPTOApp01].	
Step 22	[CRYPTOApp01] Seek decryption position of <Ciphertext2> and state of stream cipher context by <Offset2>.	
Step 23	[CRYPTOApp01] Return seek status (success/failure) to [CRYPTO Tester].	
Step 24	[CRYPTO Tester] Check seek status.	[CRYPTO Tester] Seek status contains success and no error.
Step 25	[CRYPTO Tester] Trigger remaining decryption of <Ciphertext2>.	
Step 26	[CRYPTOApp01] Resume and complete decryption of <Ciphertext2> to obtain <Plaintext2'>.	
Step 27	[CRYPTOApp01] Return decryption status (success/failure) to [CRYPTO Tester].	
Step 28	[CRYPTO Tester] Check decryption status.	[CRYPTO Tester] Decryption status contains success and no error.
Step 29	[CRYPTO Tester] Send <Plaintext2> (created in the same way as <Plaintext2'> on the [CRYPTO Tester] side) to [CRYPTOApp01] for comparison.	
Step 30	[CRYPTOApp01] Compare <Plaintext2'> with <Plaintext2>.	
Step 31	[CRYPTOApp01] Return comparison result (matched/unmatched) to [CRYPTO Tester].	
Step 32	[CRYPTO Tester] Check comparison result.	[CRYPTO Tester] Comparison result is "matched."

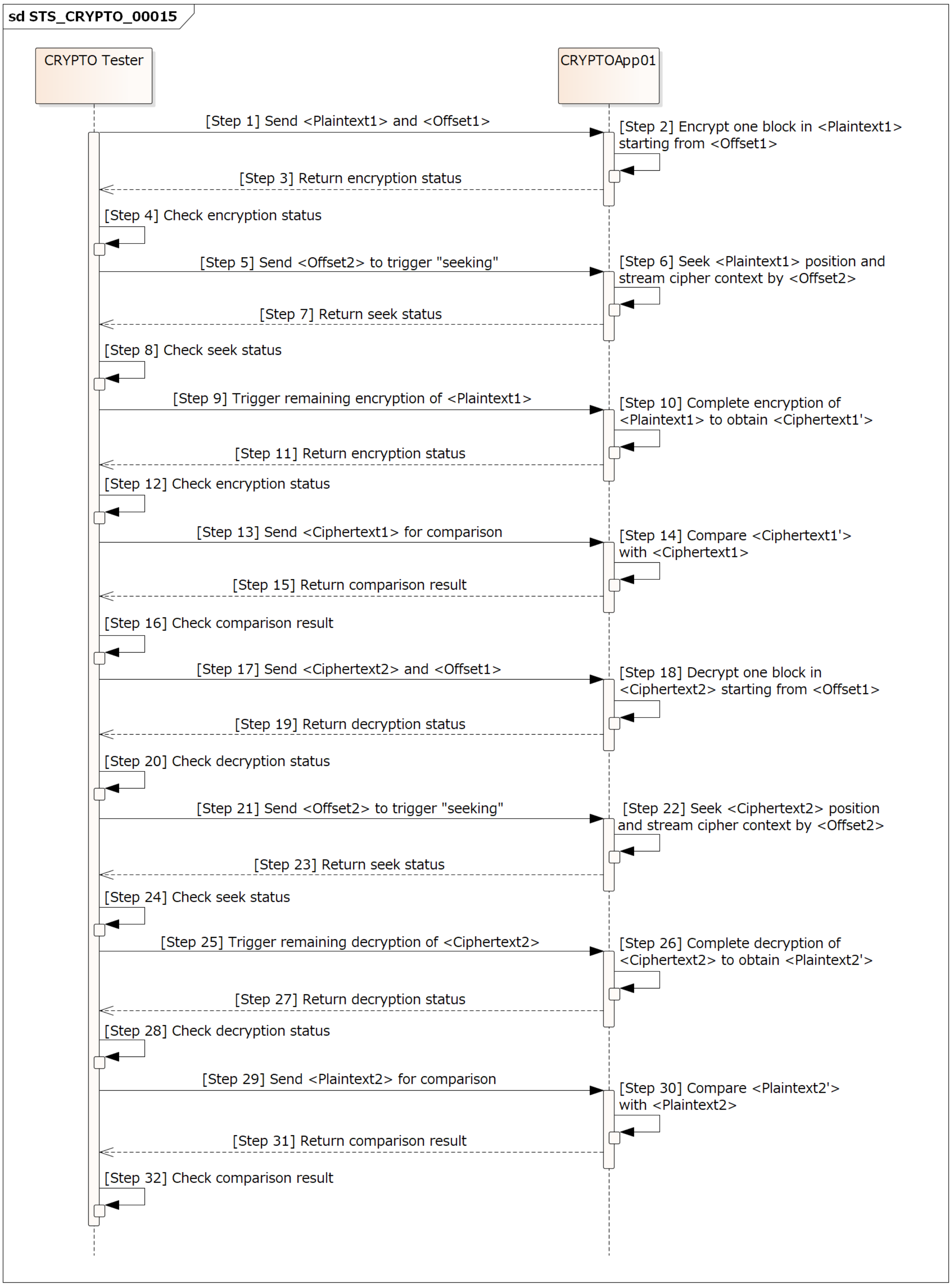
sd STS_CRYPTO_00015
CRYPTO Tester
CRYPTOApp01
[Step 1] Send <Plaintext1> and <Offset1>
[Step 2] Encrypt one block in <Plaintext1> starting from <Offset1>
[Step 3] Return encryption status
[Step 4] Check encryption status
[Step 5] Send <Offset2> to trigger "seeking"
[Step 6] Seek <Plaintext1> position and stream cipher context by <Offset2>
[Step 7] Return seek status
[Step 8] Check seek status
[Step 9] Trigger remaining encryption of <Plaintext1>
[Step 10] Complete encryption of <Plaintext1> to obtain <Ciphertext1'>
[Step 11] Return encryption status
[Step 12] Check encryption status
[Step 13] Send <Ciphertext1> for comparison
[Step 14] Compare <Ciphertext1'> with <Ciphertext1>
[Step 15] Return comparison result
[Step 16] Check comparison result
[Step 17] Send <Ciphertext2> and <Offset1>
[Step 18] Decrypt one block in <Ciphertext2> starting from <Offset1>
[Step 19] Return decryption status
[Step 20] Check decryption status
[Step 21] Send <Offset2> to trigger "seeking"
[Step 22] Seek <Ciphertext2> position and stream cipher context by <Offset2>
[Step 23] Return seek status
[Step 24] Check seek status
[Step 25] Trigger remaining decryption of <Ciphertext2>
[Step 26] Complete decryption of <Ciphertext2> to obtain <Plaintext2'>
[Step 27] Return decryption status
[Step 28] Check decryption status
[Step 29] Send <Plaintext2> for comparison
[Step 30] Compare <Plaintext2'> with <Plaintext2>
[Step 31] Return comparison result
[Step 32] Check comparison result

Figure 15.14: Sequence diagram of STS_CRYPTO_00015.

15.2.16 [STS_CRYPTO_00016] Identification and version control of cryptographic objects and key slots.

Test Objective	Verify that Crypto Stack correctly handles UUIDs and versions of cryptographic objects and key slots.		
ID	STS_CRYPTO_00016	**State**	Draft
Affected Functional Cluster	Cryptography		
Trace to RS Criteria	[RS_CRYPTO_02005], [RS_CRYPTO_02006], [RS_CRYPTO_02116], [RS_CRYPTO_02110], [RS_CRYPTO_02405]		
Reference to Test Environment	STC_CRYPTO_00001 in Test configurations		
Configuration Parameters	- Configure [CRYPTO Tester] to have common symmetric keys <SK1>, <SK2> and <SK3> with: – COUIDs <SK1UID>, <SK2UID>, and <SK3UID>, respectively (each containing a generator (origin) UUID and a version stamp). – a common generator UUID for <SK1>, <SK2>, and <SK3>. – <SK1>'s version stamp <SK1Ver> earlier than <SK2>'s version stamp <SK2Ver>, and <SK2Ver> ealier than <SK3>'s version stamp <SK3Ver>. - Configure [CRYPTO Tester] to have instance specifier <KeySlot1IS> of <KeySlot1>. - Configure [CRYPTOApp01] to have a key slot <KeySlot1> with <SK2> already saved, and with following prototyped properties (<KeySlot1PProp>): – version stamp later than <SK1Ver> and earlier than <SK3Ver>. – max number of allowed updates = 1. – enough slot capacity to save <SK1>/<SK2>/<SK3>. - Configure [CRYPTOApp01] as the owner of <SK1>, <SK2>, <SK3>, and <KeySlot1>.		
Summary	[CRYPTO Tester] checks whether [CRYPTOApp01] correctly: - loads key slot <KeySlot1> by its instance specifier <KeySlot1IS>. - retrieves prototyped properties <KeySlot1PProp> of <KeySlot1>. - loads cryptographic object <SK2> from <KeySlot1>. - retrieves COUID from cryptographic object <SK2>. - compares versions between two cryptographic objects <SK1>, <SK2>, and <SK3>.		
Pre-conditions	- Communication between [CRYPTO Tester] and [CRYPTOApp01] has been set up. - Symmetric keys <SK1>, <SK2>, and <SK3> can be accessed by [CRYPTOApp01].		
Post-conditions	Communication between [CRYPTO Tester] and [CRYPTOApp01] is closed.		
Main Test Execution			
Test Steps		**Pass Criteria**	
Step 1	[CRYPTO Tester] Trigger loading key slot <KeySlot1> by sending <KeySlot1>'s instance specifier <KeySlot1IS> to [CRYPTOApp01].		
Step 2	[CRYPTOApp01] Load <KeySlot1> by passing <KeySlot1IS> to KeyStorageProvider::LoadKeySlot API.		
Step 3	[CRYPTOApp01] Return status of loading <KeySlot1> to [CRYPTO Tester].		
Step 4	[CRYPTO Tester] Check status of loading <KeySlot1>.	[CRYPTO Tester] Status contains success and no error.	
Step 5	[CRYPTO Tester] Trigger retrieving prototyped properties <KeySlot1PProp'> of key slot <KeySlot1>.		

▽

△

Step 6	[CRYPTOApp01] Retrieve prototyped properties of <KeySlot1> to obtain <KeySlot1PProp'>.	
Step 7	[CRYPTOApp01] Return status (success/failure) of retrieving <KeySlot1PProp'> to [CRYPTO Tester].	
Step 8	[CRYPTO Tester] Check status of retrieving <KeySlot1PProp'>.	[CRYPTO Tester] Status contains success and no error.
Step 9	[CRYPTO Tester] Send <KeySlot1PProp1> to [CRYPTOApp01] for comparison.	
Step 10	[CRYPTOApp01] Compare <KeySlotPProp1'> with <KeySlotPProp1>.	
Step 11	[CRYPTOApp01] Return comparison result (matched/unmatched) to [CRYPTO Tester].	
Step 12	[CRYPTO Tester] Check comparison result.	[CRYPTO Tester] Comparison result is "matched."
Step 13	[CRYPTO Tester] Send trigger of loading <SK2> to [CRYPTOApp01].	
Step 14	[CRYPTOApp01] Load <SK2> from <KeySlot1>.	
Step 15	[CRYPTOApp01] Return status (success/failure) of loading <SK2> to [CRYPTO Tester].	
Step 16	[CRYPTO Tester] Check status of loading <SK2>.	[CRYPTO Tester] Status contains success and no error.
Step 17	[CRYPTO Tester] Send trigger of retrieving COUID of loaded <SK2> to [CRYPTOApp01].	
Step 18	[CRYPTOApp01] Retrieve <SK2UID'> from loaded <SK2>.	
Step 19	[CRYPTOApp01] Return status of retrieving <SK2UID'> to [CRYPTO Tester].	
Step 20	[CRYPTO Tester] Check status of retrieving <SK2UID'>.	[CRYPTO Tester] Status contains success and no error.
Step 21	[CRYPTO Tester] Send <SK2UID> to [CRYPTOApp01] for comparison.	
Step 22	[CRYPTOApp01] Compare <SK2UID'> with <SK2UID>.	
Step 23	[CRYPTOApp01] Return comparison result (match/unmatch) to [CRYPTO Tester].	

▽

△

Step 24	[CRYPTO Tester] Check comparison result.	[CRYPTO Tester] Comparison result is "matched."
Step 25	[CRYPTO Tester] Trigger comparing <SK1>'s version stamp with <SK2>'s, and <SK2>'s version stamp with <SK3>'s by sending <SK1> and <SK3> (in exported format) to [CRYPTOApp01].	
Step 26	[CRYPTOApp01] Retrieve bool values <SK1EarlierThanSK2>, <SK1LaterThanSK2>, <SK2EarlierThanSK3>, and <SK2LaterThanSK3> by calling APIs "HasEarlierVersionThan" and "HasLaterVersionThan".	
Step 27	[CRYPTOApp01] Return <SK1EarlierThanSK2>, <SK1LaterThanSK2>, <SK2EarlierThanSK3>, and <SK2LaterThanSK3> to [CRYPTO Tester].	
Step 28	[CRYPTO Tester] Check values of <SK1EarlierThanSK2>, <SK1LaterThanSK2>, <SK2EarlierThanSK3>, and <SK2LaterThanSK3>.	[CRYPTO Tester] <SK1EarlierThanSK2> matches "true", <SK1LaterThanSK2> matches "false", <SK2EarlierThanSK3> matches "true", and <SK2LaterThanSK3> matches "false".

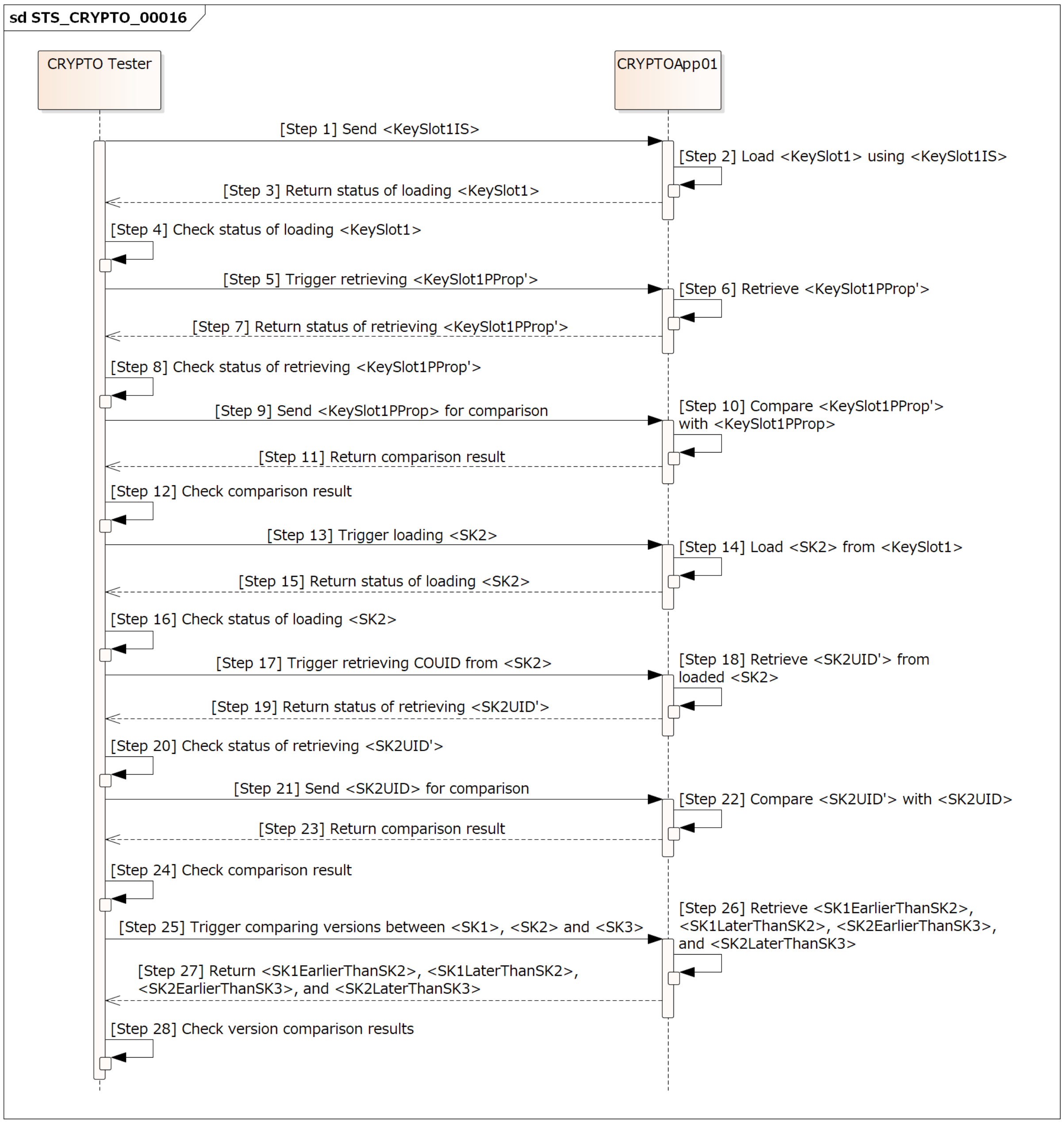
sd STS_CRYPTO_00016
CRYPTO Tester
CRYPTOApp01
[Step 1] Send <KeySlot1IS>
[Step 2] Load <KeySlot1> using <KeySlot1IS>
[Step 3] Return status of loading <KeySlot1>
[Step 4] Check status of loading <KeySlot1>
[Step 5] Trigger retrieving <KeySlot1PProp'>
[Step 6] Retrieve <KeySlot1PProp'>
[Step 7] Return status of retrieving <KeySlot1PProp'>
[Step 8] Check status of retrieving <KeySlot1PProp'>
[Step 9] Send <KeySlot1PProp> for comparison
[Step 10] Compare <KeySlot1PProp'> with <KeySlot1PProp>
[Step 11] Return comparison result
[Step 12] Check comparison result
[Step 13] Trigger loading <SK2>
[Step 14] Load <SK2> from <KeySlot1>
[Step 15] Return status of loading <SK2>
[Step 16] Check status of loading <SK2>
[Step 17] Trigger retrieving COUID from <SK2>
[Step 18] Retrieve <SK2UID'> from loaded <SK2>
[Step 19] Return status of retrieving <SK2UID'>
[Step 20] Check status of retrieving <SK2UID'>
[Step 21] Send <SK2UID> for comparison
[Step 22] Compare <SK2UID'> with <SK2UID>
[Step 23] Return comparison result
[Step 24] Check comparison result
[Step 25] Trigger comparing versions between <SK1>, <SK2> and <SK3>
[Step 26] Retrieve <SK1EarlierThanSK2>, <SK1LaterThanSK2>, <SK2EarlierThanSK3>, and <SK2LaterThanSK3>
[Step 27] Return <SK1EarlierThanSK2>, <SK1LaterThanSK2>, <SK2EarlierThanSK3>, and <SK2LaterThanSK3>
[Step 28] Check version comparison results

Figure 15.15: Sequence diagram of STS_CRYPTO_00016.

15.2.17 [STS_CRYPTO_00017] Run-time properties of PrivateKey, SignerPrivateCtx, and KeyDecapsulatorPrivateCtx.

Test Objective	Verify that Crypto Stack supports querying run-time properties of PrivateKey, SignerPrivateCtx, and KeyDecapsulatorPrivateCtx.		
ID	STS_CRYPTO_00017	**State**	Draft

▽

△

Affected Functional Cluster	Cryptography	
Trace to RS Criteria	[RS_CRYPTO_02309], [RS_CRYPTO_02005]	
Reference to Test Environment	STC_CRYPTO_00001 in Test configurations	
Configuration Parameters	- [CRYPTOApp01] to have an asymmetric private key <APvK1> for algorithm <Alg1> (e.g. "SIG/ECDSA-256,SHA2-256") stored in a key slot accessible by an instance specifier <KeySlot1IS>. - [CRYPTO Tester] to have a hash algorithm <Alg2> (e.g. "SHA2-256") supported by tested Crypto implementation. Algorithms <Alg1>, <Alg2>, and their associated/expected values are mentioned as examples (in parenthesis with "e.g." notation).	
Summary	[CRYPTO Tester] checks whether [CRYPTOApp01] correctly: 1. retrieves <APvK1>'s: – payload size <APvK1Payload> and – primitive ID of used algorithm <APvK1AlgId>. 2. retrieves <SigPvCtx1>'s: – required hash size <SigPvCtx1ReqHashSize>, – required hash algorithm ID <SigPvCtx1ReqHashAlgId>, – signature size <SigPvCtx1SigSize>, – initialization status flag <SigPvCtx1Init>, – actual key bit length <SigPvCtx1ActKeyLen>, – actual key COUID <SigPvCtx1KeyUID>, – key available flag <SigPvCtx1KeyAvailable>, – minimum key bit length <SigPvCtx1MinKeyLen>, – maximum key bit length <SigPvCtx1MaxKeyLen>, – and a key bit length <APvK1Len> (e.g. 256) support flag <SigPvCtx1KeyLenSupport>. 3. retrieves <HFCtx1>'s: – digest size <HFCtx1DigestSize>, – processing started flag <HFCtx1Started>, – and processing finished flag <HFCtx1Finished>.	
Pre-conditions	- Communication between [CRYPTO Tester] and [CRYPTOApp01] has been set up.	
Post-conditions	Communication between [CRYPTO Tester] and [CRYPTOApp01] is closed.	
Main Test Execution		
Test Steps		**Pass Criteria**
Step 1	[CRYPTO Tester] Trigger loading asymmetric private key <APvK1> by sending <KeySlot1IS> to [CRYPTOApp01].	
Step 2	[CRYPTOApp01] Load <APvK1> using <APvK1IS>.	
Step 3	[CRYPTOApp01] Return status (success/failure) of loading <APvK1> to [CRYPTO Tester].	
Step 4	[CRYPTO Tester] Check status of loading <APvK1>.	[CRYPTO Tester] Status contains success and no error.
Step 5	[CRYPTO Tester] Trigger retrieving <APvK1>'s payload size <APvK1Payload> and primitive ID <APvK1AlgId>.	
Step 6	[CRYPTOApp01] Retrieve and return values of <APvK1Payload> and <APvK1AlgId> to [CRYPTO Tester].	

▽

△

Step 7	[CRYPTO Tester] Check values of <APvK1Payload> and <APvK1AlgId>.	[CRYPTO Tester] <APvK1Payload> matches expected size by implementation. <APvKAlgId> matches expected algorithm (e.g. "SIG/ECDSA-256,SHA-256").
Step 8	[CRYPTO Tester] Send trigger of creating SignerPrivateCtx <SigPvCtx1> to [CRYPTOApp01].	
Step 9	[CRYPTOApp01] Create <SigPvCtx1> using <APvK1AlgId>.	
Step 10	[CRYPTOApp01] Return status (success/failure) of creating <SigPvCtx1> to [CRYPTO Tester].	
Step 11	[CRYPTO Tester] Check status of creating <SigPvCtx1>.	[CRYPTO Tester] Status contains success and no error.
Step 12	[CRYPTO Tester] Trigger retrieving <SigPvCtx1>'s required hash size <SigPvCtx1ReqHashSize>, required hash alogrithm ID <SigPvCtx1ReqHashAlgId>, and signature size <SigPvCtx1SigSize>.	
Step 13	[CRYPTOApp01] Retrieve and return values of <SigPvCtx1ReqHashSize>, <SigPvCtx1ReqHashAlgId>, and <SigPvCtx1SigSize> to [CRYPTO Tester].	
Step 14	[CRYPTO Tester] Check values of <SigPvCtx1ReqHashSize>, <SigPvCtx1ReqHashAlgId>, and <SigPvCtx1SigSize>.	[CRYPTO Tester] <SigPvCtx1ReqHashSize> matches 32. <SigPvCtx1ReqHashAlgId> matches expected hash algorithm (e.g. "SHA-256"). <SigPvCtx1SigSize> matches 32.
Step 15	[CRYPTO Tester] Trigger retrieving <SigPvCtx1>'s initialization flag <SigPvCtx1Init>, actual key bit length <SigPvCtx1ActKeyLen>, actual key COUID <SigPvCtx1KeyUID>, key available flag <SigPvCtx1KeyAvailable>, minimum key bit length <SigPvCtx1MinKeyLen>, maximum key bit length <SigPvCtx1MaxKeyLen>, and key bit length <APvK1Len> (e.g. 256) support flag <SigPvCtx1KeyLenSupport>.	
Step 16	[CRYPTOApp01] Retrieve and return values of <SigPvCtx1Init>, <SigPvCtx1ActKeyLen>, <SigPvCtx1KeyUID>, <SigPvCtx1KeyAvailable>, <SigPvCtx1MinKeyLen>, <SigPvCtx1MaxKeyLen>, and <SigPvCtx1KeyLenSupport> to [CRYPTO Tester].	

▽

△

Step 17	[CRYPTO Tester] Check values of <SigPvCtx1Init>, <SigPvCtx1ActKeyLen>, <SigPvCtx1KeyUID>, <SigPvCtx1KeyAvailable>, <SigPvCtx1MinKeyLen>, <SigPvCtx1MaxKeyLen>, and <SigPvCtx1KeyLenSupport>.	[CRYPTO Tester] <SigPvCtx1Init> matches false. <SigPvCtx1ActKeyLen> matches 0. <SigPvCtx1KeyUID> matches Nil. <SigPvCtx1KeyAvailable> matches false. <SigPvCtx1MinKeyLen> and <SigPvCtx1MinKeyLen> match expected values by implementation. <SigPvCtx1KeyLenSupport> matches true.
Step 18	[CRYPTO Tester] Trigger setting <APvK1> to <SigPvCtx1>.	
Step 19	[CRYPTOApp01] Set <APvK1> to <SigPvCtx1>.	
Step 20	[CRYPTOApp01] Return status (success/failure) of setting <APvK1> to [CRYPTO Tester].	
Step 21	[CRYPTO Tester] Check status of setting <APvK1>.	[CRYPTO Tester] Status contains success and no error.
Step 22	[CRYPTO Tester] Trigger retrieving <SigPvCtx1>'s initialization flag <SigPvCtx1Init>, actual key bit length <SigPvCtx1ActKeyLen>, actual key COUID <SigPvCtx1KeyUID>, and key available flag <SigPvCtx1KeyAvailable>.	
Step 23	[CRYPTOApp01] Retrieve and return values of <SigPvCtx1Init>, <SigPvCtx1ActKeyLen>, <SigPvCtx1KeyUID>, and <SigPvCtx1KeyAvailable> to [CRYPTO Tester].	
Step 24	[CRYPTO Tester] Check values of <SigPvCtx1Init>, <SigPvCtx1ActKeyLen>, <SigPvCtx1KeyUID>, and <SigPvCtx1KeyAvailable>.	[CRYPTO Tester] <SigPvCtx1Init> matches true. <SigPvCtx1ActKeyLen> matches <APvK1Len> (e.g. 256). <SigPvCtx1KeyUID> matches <APvK1UID>. <SigPvCtx1KeyAvailable> matches true.
Step 25	[CRYPTO Tester] Trigger creating <HFCtx1> by sending <Alg2> to [CRYPTOApp01].	
Step 26	[CRYPTOApp01] Create <HFCtx1> using <Alg2>.	
Step 27	[CRYPTOApp01] Return status (success/failure) of creating <HFCtx1> to [CRYPTO Tester].	
Step 28	[CRYPTO Tester] Check status of creating <HFCtx1>.	[CRYPTO Tester] Status contains success and no error.
Step 29	[CRYPTO Tester] Trigger retrieving <HFCtx1>'s digest size <HFCtx1DigestSize>, started flag <HFCtx1Started>, and finished flag <HFCtx1Finished>.	

▽

△

Step 30	[CRYPTOApp01] Retrieve and return values of <HFCtx1DigestSize>, <HFCtx1Started>, and <HFCtx1Finished> to [CRYPTO Tester].	
Step 31	[CRYPTO Tester] Check values of <HFCtx1DigestSize>, <HFCtx1Started>, and <HFCtx1Finished>.	[CRYPTO Tester] <HFCtx1DigestSize> matches 32. <HFCtx1Started> matches false. <HFCtx1Finished> matches false.
Step 32	[CRYPTO Tester] Send trigger of calling Start method to [CRYPTOApp01].	
Step 33	[CRYPTOApp01] Call Start method of <HFCtx1>.	
Step 34	[CRYPTOApp01] Return call status (success/failure) of Start method to [CRYPTO Tester].	
Step 35	[CRYPTO Tester] Check call status of Start method.	[CRYPTO Tester] Status contains success and no error.
Step 36	[CRYPTO Tester] Trigger retrieving <HFCtx1>'s started flag <HFCtx1Started> and finished flag <HFCtx1Finished>.	
Step 37	[CRYPTOApp01] Retrieve and return values of <HFCtx1Started>, and <HFCtx1Finished> to [CRYPTO Tester].	
Step 38	[CRYPTO Tester] Check values of <HFCtx1Started>, and <HFCtx1Finished>.	[CRYPTO Tester] <HFCtx1Started> matches true. <HFCtx1Finished> matches false.
Step 39	[CRYPTO Tester] Send trigger of calling Update and Finish methods to [CRYPTOApp01].	
Step 40	[CRYPTOApp01] Call Update method of <HFCtx1> with arbitrary data, and then call Finish method of <HFCtx1>.	
Step 41	[CRYPTOApp01] Return call status (success/failure) of Update and Finish methods to [CRYPTO Tester].	
Step 42	[CRYPTO Tester] Check call statuses of Update and Finish methods.	[CRYPTO Tester] Statuses contain success and no error.
Step 43	[CRYPTO Tester] Trigger retrieving <HFCtx1>'s started flag <HFCtx1Started> and finished flag <HFCtx1Finished>.	
Step 44	[CRYPTOApp01] Retrieve and return values of <HFCtx1Started>, and <HFCtx1Finished> to [CRYPTO Tester].	
Step 45	[CRYPTO Tester] Check values of <HFCtx1Started>, and <HFCtx1Finished>.	[CRYPTO Tester] <HFCtx1Started> matches false. <HFCtx1Finished> matches true.

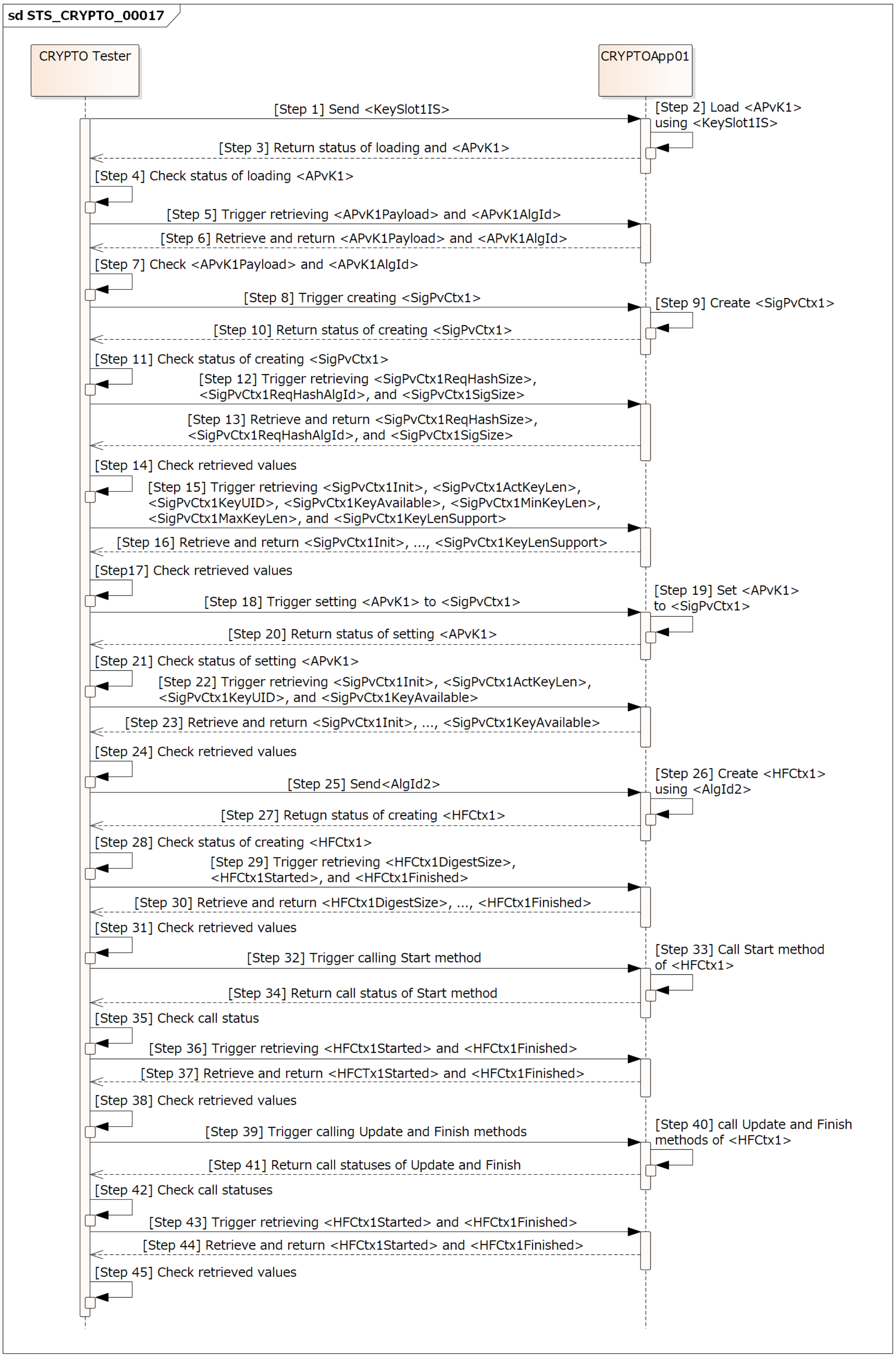
sd STS_CRYPTO_00017
CRYPTO Tester
CRYPTOApp01
[Step 1] Send <KeySlot1IS>
[Step 2] Load <APvK1> using <KeySlot1IS>
[Step 3] Return status of loading and <APvK1>
[Step 4] Check status of loading <APvK1>
[Step 5] Trigger retrieving <APvK1Payload> and <APvK1AlgId>
[Step 6] Retrieve and return <APvK1Payload> and <APvK1AlgId>
[Step 7] Check <APvK1Payload> and <APvK1AlgId>
[Step 8] Trigger creating <SigPvCtx1>
[Step 9] Create <SigPvCtx1>
[Step 10] Return status of creating <SigPvCtx1>
[Step 11] Check status of creating <SigPvCtx1>
[Step 12] Trigger retrieving <SigPvCtx1ReqHashSize>, <SigPvCtx1ReqHashAlgId>, and <SigPvCtx1SigSize>
[Step 13] Retrieve and return <SigPvCtx1ReqHashSize>, <SigPvCtx1ReqHashAlgId>, and <SigPvCtx1SigSize>
[Step 14] Check retrieved values
[Step 15] Trigger retrieving <SigPvCtx1Init>, <SigPvCtx1ActKeyLen>, <SigPvCtx1KeyUID>, <SigPvCtx1KeyAvailable>, <SigPvCtx1MinKeyLen>, <SigPvCtx1MaxKeyLen>, and <SigPvCtx1KeyLenSupport>
[Step 16] Retrieve and return <SigPvCtx1Init>, ..., <SigPvCtx1KeyLenSupport>
[Step17] Check retrieved values
[Step 18] Trigger setting <APvK1> to <SigPvCtx1>
[Step 19] Set <APvK1> to <SigPvCtx1>
[Step 20] Return status of setting <APvK1>
[Step 21] Check status of setting <APvK1>
[Step 22] Trigger retrieving <SigPvCtx1Init>, <SigPvCtx1ActKeyLen>, <SigPvCtx1KeyUID>, and <SigPvCtx1KeyAvailable>
[Step 23] Retrieve and return <SigPvCtx1Init>, ..., <SigPvCtx1KeyAvailable>
[Step 24] Check retrieved values
[Step 25] Send<AlgId2>
[Step 26] Create <HFCtx1> using <AlgId2>
[Step 27] Retugn status of creating <HFCtx1>
[Step 28] Check status of creating <HFCtx1>
[Step 29] Trigger retrieving <HFCtx1DigestSize>, <HFCtx1Started>, and <HFCtx1Finished>
[Step 30] Retrieve and return <HFCtx1DigestSize>, ..., <HFCtx1Finished>
[Step 31] Check retrieved values
[Step 32] Trigger calling Start method
[Step 33] Call Start method of <HFCtx1>
[Step 34] Return call status of Start method
[Step 35] Check call status
[Step 36] Trigger retrieving <HFCtx1Started> and <HFCtx1Finished>
[Step 37] Retrieve and return <HFCTx1Started> and <HFCtx1Finished>
[Step 38] Check retrieved values
[Step 39] Trigger calling Update and Finish methods
[Step 40] call Update and Finish methods of <HFCtx1>
[Step 41] Return call statuses of Update and Finish
[Step 42] Check call statuses
[Step 43] Trigger retrieving <HFCtx1Started> and <HFCtx1Finished>
[Step 44] Retrieve and return <HFCtx1Started> and <HFCtx1Finished>
[Step 45] Check retrieved values

Figure 15.16: Sequence diagram of STS_CRYPTO_00017.

15.2.18 [STS_CRYPTO_00018] Run-time properties of cryptographic primitives - SymmetricBlockCipherCtx, AuthCipherCtx, and KeyDecapsulatorPrivateCtx.

Test Objective	Verify that Crypto Stack supports querying run-time properties of cryptographic primitives SymmetricBlockCipherCtx, AuthCipherCtx, and KeyDecapsulatorPrivateCtx.		
ID	STS_CRYPTO_00018	**State**	Draft
Affected Functional Cluster	Cryptography		
Trace to RS Criteria	[RS_CRYPTO_02309], [RS_CRYPTO_02205]		
Reference to Test Environment	STC_CRYPTO_00001 in Test configurations		
Configuration Parameters	- [CRYPTOApp01] to have a symmetric key <SK1> for algorithm <Alg1> (e.g. "GCM/AES-128") stored in a key slot accessible by an instance specifier <KeySlot1IS>. - [CRYPTO Tester] to have <SK1>'s associated algorithm information <Alg1> (e.g. "GCM/AES-128"). - [CRYPTO Tester] to have <Alg2> (e.g. "KEM/RSA-2048") for KeyDecapsulatorPrivateCtx. Algorithms <Alg1>, <Alg2>, and their associated/expected values are mentioned as examples (in parenthesis with "e.g." notation).		
Summary	[CRYPTO Tester] checks whether [CRYPTOApp01] correctly: 1. retrieves SymmetricBlockCipherCtx <SymBCCtx1>'s: – block size <SymBCCtx1BlkSize>, – kind of transformation <SymBCCtx1Transform>, – max input only flag <SymBCCtx1MaxIOnly>, – max output only flag <SymBCCtx1MaxOOnly>, – maximum input size <SymBCCtx1MaxISize>, – and maximum output size <SymBCCtx1MaxOSize>. 2. retrieves AuthCipherCtx <AuthCCtx1>'s: – maximum associated data size <AuthCCtxMaxDataSize>, – IV size <AuthCCtx1IVSize>, – block size <AuthCCtx1BlkSize>, – validity flag <AuthCCtx1ValidIVSize> of an IV size <Alg1IvSize> (e.g. 16), – and actual IV bit length <AuthCCtx1ActIvLen>. 3. retrieves StreamCipherCtx <StrCCtx1>'s: – kind of transformation <StrCCtx1Transform>, – byte-wise mode flag <StrCCtx1ByteMode>, – and seekable mode flag <StrCCtx1SeekableMode>. 4. retrieves KeyDecapsulatorPrivateCtx <KeyDecPvCtx1>'s: – KEK entropy <KeyDecPvCtx1KekEnt>, – and encapsulated size <KeyDecPvCtx1EncSize>.		
Pre-conditions	- Communication between [CRYPTO Tester] and [CRYPTOApp01] has been set up.		
Post-conditions	Communication between [CRYPTO Tester] and [CRYPTOApp01] is closed.		
Main Test Execution			
Test Steps		**Pass Criteria**	
Step 1	[CRYPTO Tester] Trigger loading symmetric key <SK1> by sending <KeySlot1IS>.		
Step 2	[CRYPTOApp01] Load <SK1> using <KeySlot1IS>.		
Step 3	[CRYPTOApp01] Return status (success/failure) of loading <SK1> to [CRYPTO Tester].		
Step 4	[CRYPTO Tester] Check status of loading <SK1>.	[CRYPTO Tester] Status contains success and no error.	

▽

△

Step 5	[CRYPTO Tester] Trigger creating SymmetricBlockCipherCtx <SymBCCtx1>.	
Step 6	[CRYPTOApp01] Create <SymBCCtx1> using <Alg1> retrieved from <SK1>.	
Step 7	[CRYPTOApp01] Return status (success/failure) of creating <SymBCCtx1> to [CRYPTO Tester].	
Step 8	[CRYPTO Tester] Check status of creating <SymBCCtx1>.	[CRYPTO Tester] Status contains success and no error.
Step 9	[CRYPTO Tester] Trigger setting <SK1> to <SymBCCtx1>.	
Step 10	[CRYPTOApp01] Set <SK1> to <SymBCCtx1> with encryption mode.	
Step 11	[CRYPTOApp01] Return status (success/failure) of setting <SK1> to [CRYPTO Tester].	
Step 12	[CRYPTO Tester] Check status of setting <SK1>.	[CRYPTO Tester] Status contains success and no error.
Step 13	[CRYPTO Tester] Trigger retrieving <SymBCCtx1>'s block size <SymBCCtx1BlkSize>, kind of transformation <SymBCCtx1Transform>, max input only flag <SymBCCtx1MaxIOnly>, max output only flag <SymBCCtx1MaxOOnly>, maximum input size <SymBCCtx1MaxISize>, and maximum output size <SymBCCtx1MaxOSize>.	
Step 14	[CRYPTOApp01] Retrieve and return values of <SymBCCtx1BlkSize>, <SymBCCtx1Transform>, <SymBCCtx1MaxIOnly>, <SymBCCtx1MaxOOnly>, <SymBCCtx1MaxISize>, and <SymBCCtx1MaxOSize> to [CRYPTO Tester].	
Step 15	[CRYPTO Tester] Check values of <SymBCCtx1BlkSize>, <SymBCCtx1Transform>, <SymBCCtx1MaxIOnly>, <SymBCCtx1MaxOOnly>, <SymBCCtx1MaxISize>, and <SymBCCtx1MaxOSize>.	[CRYPTO Tester] <SymBCCtx1BlkSize> matches value expected by <Alg1> (e.g. 16). <SymBCCtx1Transform> matches CryptoTransform::kEncrypt. <SymBCCtx1MaxIOnly> and <SymBCCtx1MaxOOnly> match value (true/false) expected by <Alg1>. <SymBCCtx1MaxISize> and <SymBCCtx1MaxOSize> match value expected by <Alg1> (e.g. 16).
Step 16	[CRYPTO Tester] Trigger creating AuthCipherCtx <AuthCCtx1> by sending <Alg1> to [CRYPTOApp01].	
Step 17	[CRYPTOApp01] Create <AuthCCtx1> using <Alg1>.	

▽

△

Step 18	[CRYPTOApp01] Return status (success/failure) of creating <AuthCCtx1> to [CRYPTO Tester].	
Step 19	[CRYPTO Tester] Check status of creating <AuthCCtx1>.	[CRYPTO Tester] Status contains success and no error.
Step 20	[CRYPTO Tester] Trigger retrieving <ACCtx1>'s maximum associated data size <AuthCCtx1MaxDataSize>, IV size <AuthCCtx1IVSize>, block size <AuthCCTx1BlkSize>, validity flag <AuthCCtx1ValidIVSize> of an IV size <Alg1IvSize> (e.g. 16), and actual IV bit length <AuthCCtx1ActIvLen>.	
Step 21	[CRYPTOApp01] Retrieve and return values of <AuthCCtx1MaxDataSize>, <AuthCCtx1IVSize>, <AuthCCTx1BlkSize>, <AuthCCtx1ValidIVSize>, and <AuthCCtx1ActIvLen> to [CRYPTO Tester].	
Step 22	[CRYPTOApp01] Check values of <AuthCCtx1MaxDataSize>, <AuthCCtx1IVSize>, <AuthCCTx1BlkSize>, <AuthCCtx1ValidIVSize>, and <AuthCCtx1ActIvLen>.	[CRYPTO Tester] <AuthCCtx1MaxDataSize>, <AuthCCtx1IVSize>, and <AuthCCTx1BlkSize> match values expected by <Alg1> (e.g. 16). <AuthCCtx1ValidIVSize> matches value (true/false) expected by <Alg1IvSize> and <Alg1>. <AuthCCtx1ActIvLen> matches value expected by <Alg1> (e.g. 96).
Step 23	[CRYPTO Tester] Trigger creating StreamCipherCtx <StrCCtx1>.	
Step 24	[CRYPTOApp01] Create <StrCCtx1> using <Alg1> retrieved from <SK1>.	
Step 25	[CRYPTOApp01] Return status (success/failure) of creating <StrCCtx1> to [CRYPTO Tester].	
Step 26	[CRYPTO Tester] Check status of creating <StrCCtx1>.	[CRYPTO Tester] Status contains success and no error.
Step 27	[CRYPTO Tester] Trigger setting <SK1> to <StrCCtx1>.	
Step 28	[CRYPTOApp01] Set <SK1> to <StrCCtx1> with decryption mode.	
Step 29	[CRYPTOApp01] Return status (success/failure) of setting <SK1> to [CRYPTO Tester].	
Step 30	[CRYPTO Tester] Check status of setting <SK1>.	[CRYPTO Tester] Status contains success and no error.
Step 31	[CRYPTO Tester] Trigger retrieving <StrCCtx1>'s kind of transformation <StrCCtx1Transform>, byte-wise mode flag <StrCCtx1ByteMode>, and seekable mode flag <StrCCtx1SeekableMode>.	

▽

△

Step 32	[CRYPTOApp01] Retrieve and return values of <StrCCtx1Transform>, <StrCCtx1ByteMode>, and <StrCCtx1SeekableMode> to [CRYPTO Tester].	
Step 33	[CRYPTO Tester] Check values of <StrCCtx1Transform>, <StrCCtx1ByteMode>, and <StrCCtx1SeekableMode>.	[CRYPTO Tester] <StrCCtx1Transform> matches CryptoTransform::kDecrypt. <StrCCtx1ByteMode> matches value expected by <Alg1> (e.g. false). <StrCCtx1SeekableMode> matches value expected by <Alg1> (e.g. true).
Step 34	[CRYPTO Tester] Trigger creating KeyDecapsulatorPrivateCtx <KeyDecPvCtx1> by sending <Alg2> to [CRYPTOApp01].	
Step 35	[CRYPTOApp01] Create <KeyDecPvCtx1> using <Alg2>.	
Step 36	[CRYPTOApp01] Return status (success/failure) of creating <KeyDecPvCtx1> to [CRYPTO Tester]	
Step 37	[CRYPTO Tester] Check status of creating <KeyDecPvCtx1>.	[CRYPTO Tester] Status contains success and no error.
Step 38	[CRYPTO Tester] Trigger retrieving <KeyDecPvCtx1>'s KEK entropy <KeyDecPvCtx1KekEnt> and encapsulated size <KeyDecPvCtx1EncSize>.	
Step 39	[CRYPTOApp01] Retrieve and return values of <KeyDecPvCtx1KekEnt> and <KeyDecPvCtx1EncSize> to [CRYPTO Tester].	
Step 40	[CRYPTO Tester] Check values of <KeyDecPvCtx1KekEnt> and <KeyDecPvCtx1EncSize>.	[CRYPTO Tester] <KeyDecPvCtx1KekEnt> matches value expected by <Alg2>. <KeyDecPvCtx1EncSize> matches value expected by <Alg2> and implementation.

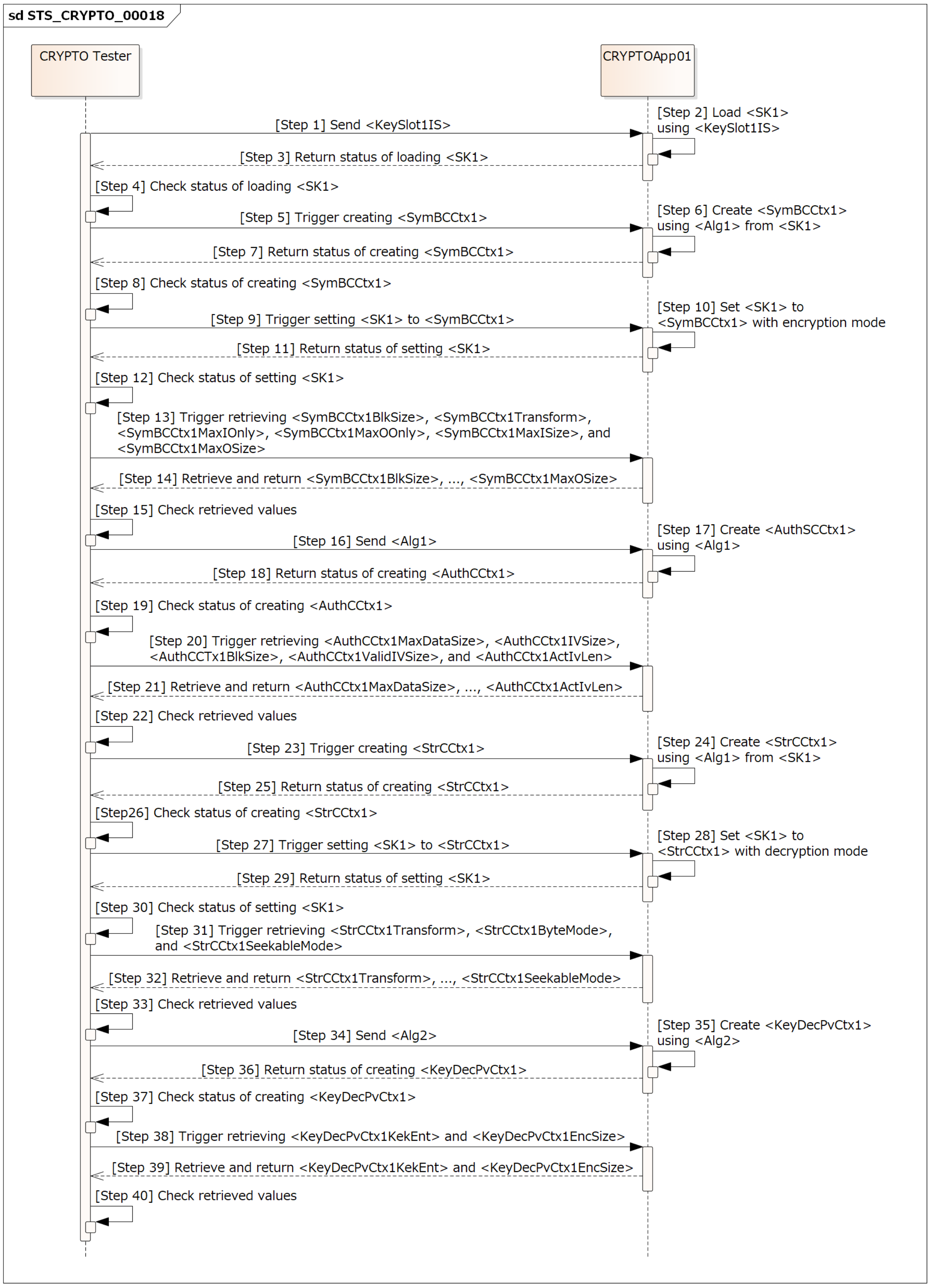
sd STS_CRYPTO_00018
CRYPTO Tester
CRYPTOApp01
[Step 1] Send <KeySlot1IS>
[Step 2] Load <SK1> using <KeySlot1IS>
[Step 3] Return status of loading <SK1>
[Step 4] Check status of loading <SK1>
[Step 5] Trigger creating <SymBCCtx1>
[Step 6] Create <SymBCCtx1> using <Alg1> from <SK1>
[Step 7] Return status of creating <SymBCCtx1>
[Step 8] Check status of creating <SymBCCtx1>
[Step 9] Trigger setting <SK1> to <SymBCCtx1>
[Step 10] Set <SK1> to <SymBCCtx1> with encryption mode
[Step 11] Return status of setting <SK1>
[Step 12] Check status of setting <SK1>
[Step 13] Trigger retrieving <SymBCCtx1BlkSize>, <SymBCCtx1Transform>, <SymBCCtx1MaxIOnly>, <SymBCCtx1MaxOOnly>, <SymBCCtx1MaxISize>, and <SymBCCtx1MaxOSize>
[Step 14] Retrieve and return <SymBCCtx1BlkSize>, ..., <SymBCCtx1MaxOSize>
[Step 15] Check retrieved values
[Step 16] Send <Alg1>
[Step 17] Create <AuthSCCtx1> using <Alg1>
[Step 18] Return status of creating <AuthCCtx1>
[Step 19] Check status of creating <AuthCCtx1>
[Step 20] Trigger retrieving <AuthCCtx1MaxDataSize>, <AuthCCtx1IVSize>, <AuthCCTx1BlkSize>, <AuthCCtx1ValidIVSize>, and <AuthCCtx1ActIvLen>
[Step 21] Retrieve and return <AuthCCtx1MaxDataSize>, ..., <AuthCCtx1ActIvLen>
[Step 22] Check retrieved values
[Step 23] Trigger creating <StrCCtx1>
[Step 24] Create <StrCCtx1> using <Alg1> from <SK1>
[Step 25] Return status of creating <StrCCtx1>
[Step26] Check status of creating <StrCCtx1>
[Step 27] Trigger setting <SK1> to <StrCCtx1>
[Step 28] Set <SK1> to <StrCCtx1> with decryption mode
[Step 29] Return status of setting <SK1>
[Step 30] Check status of setting <SK1>
[Step 31] Trigger retrieving <StrCCtx1Transform>, <StrCCtx1ByteMode>, and <StrCCtx1SeekableMode>
[Step 32] Retrieve and return <StrCCtx1Transform>, ..., <StrCCtx1SeekableMode>
[Step 33] Check retrieved values
[Step 34] Send <Alg2>
[Step 35] Create <KeyDecPvCtx1> using <Alg2>
[Step 36] Return status of creating <KeyDecPvCtx1>
[Step 37] Check status of creating <KeyDecPvCtx1>
[Step 38] Trigger retrieving <KeyDecPvCtx1KekEnt> and <KeyDecPvCtx1EncSize>
[Step 39] Retrieve and return <KeyDecPvCtx1KekEnt> and <KeyDecPvCtx1EncSize>
[Step 40] Check retrieved values

Figure 15.17: Sequence diagram of STS_CRYPTO_00018.

16 Test configuration and test steps for Platform Health Management

16.1 Test System

16.1.1 Test configurations of Health Monitoring

Configuration ID	STC_PHM_00001
Description	Standard Jenkins server for PHM Management test
ECU 2	Hardware, 192.168.100.5
Jenkins	Jenkins Server, 192.168.100.10

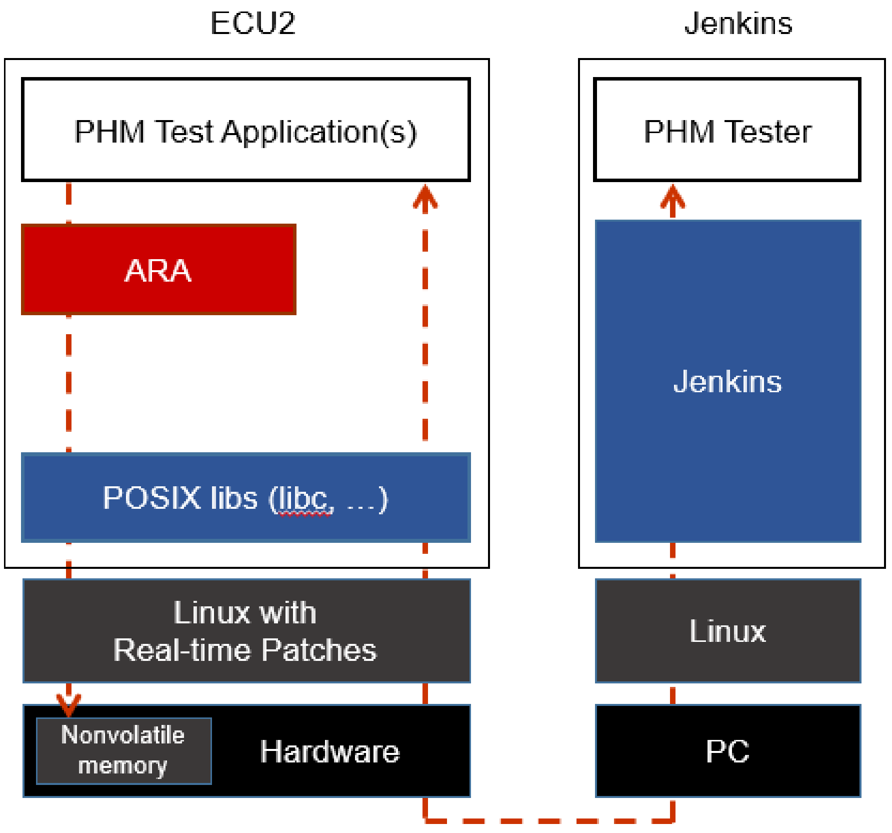

Figure 16.1: Illustration of test setup for STC-PHM-00001.

The Jenkins Server, running the job with the PHM Management test (PHM Tester) isconnected via Ethernet to ECU1 hosting the System Test Applications [PHMApp01], [PHMApp02], [PHMApp03], [PHMAppCheck].

Each application runs the corresponding supervised entities 1, 2 and 3.

The PHM Tester is supposed to check the pass criteria.

The communication between PHM Tester and the applications on the ECU may take place over the SE functional cluster in form of Application and Services messages.

16.2 Test cases

16.2.1 [STS_HM_00001] HM Performing Alive Supervision

Test Objective	Verification that the PHM management functional cluster can perform Alive Supervision and do the configured recovery actions		
ID	STS_HM_00001	**State**	Draft
Affected Functional Cluster	PHM		
Trace to RS Criteria	[RS_HM_09125]		
Reference to Test Environment	STC_PHM_00001 in Test configurations of Health Monitoring		
Configuration Parameters	- Configuring (per application), for PHMApp01, PHMApp02, PHMApp03: -AliveReferenceCycle1, AliveReferenceCycle2, AliveRefere nceCycl3 -ExpectedAliveIndications1, ExpectedAliveIndications2, E xpectedAliveIndications3 -MaxMargin1, MaxMargin2, MaxMargin3 -MinMargin1, MinMargin2, MinMargin3 -ExpiredSupervisionCyclesTolerance1, ExpiredSupervisionC yclesTolerance2, ExpiredSupervisionCyclesTolerance3 -ApplicationRecoveryAction is <Reset the Process> - Configured Manifest with Platform Health management - Machine state shall be Driving, in which all System Test Applications shall start.		
Summary	-Health Monitoring shall examine the alive supervision of 3 cyclic supervised entities. They shall report their checkpoints at the proper timing, within the configured margins. -Then after enough time, application [PHMApp02] shall miss some checkpoints reporting, yet it was for a short time, that the supervised entity went to failed state bu t not expired. -Then after another enough time, the application [PHMApp 03] shall miss some checkpoints reporting, to the extent that the supervised entity went to expired, leading to process reset.		
Pre-conditions	- PHM Tester is connected to ECU via TCP. - Software components on ECU are initialized. - ECU is in Machine State Startup and - Operating system on ECU has booted.		
Post-conditions	TCP connection between PHM Tester and ECU is closed.		
Main Test Execution			
Test Steps		**Pass Criteria**	
Step 1	[PHMApp01], [PHMApp02], [PHMApp03] All applications are reporting their checkpoints with the correct timing. i.e. reporting <ExpectedAliveIndications> within time corresponding to their <AliveReferenceCycle>.		
Step 2	[PHMAppCheck] After time corresponding to 100 x the longest <AliveReferenceCycle>, check the status of the 3 supervised entities using their Health channels.	-Health status of the 3 supervised entities is kOK	

▽

△

Step 3	[PHMApp02] Supervised entity 2 is missing some of its checkpoints, yet within its configured margins <MaxMargin2> and <MinMargin2>	
Step 4	[PHMAppCheck] After time corresponding to 100 x the longest <AliveReferenceCycle>, check the status of the 3 supervised entities using their Health channels.	-Health status of the 3 supervised entities is kOK
Step 5	[PHMApp02] Supervised entity 2 is missing some of its checkpoints, and surpassing its configured margins <MaxMargin2> and < MinMargin2>, yet for time less than <ExpiredSupervisionC yclesTolerance2>	
Step 6	[PHMAppCheck] After time corresponding to 100 x the longest <AliveRefe renceCycle>, check the status of the 3 supervised entities using their Health channels.	-Health status of the supervised entities 1 and 3 is kOK -Health status of the supervised entity 2 is kFailed
Step 7	[PHMApp03]. Supervised entity 3 is missing some of its checkpoints, and surpassing its configured margins <MaxMargin3> and < MinMargin3>, and for time more than <ExpiredSupervisionC yclesTolerance3>.	
Step 8	[PHMAppCheck]. After time corresponding to <ExpiredSupervisionCyclesTolerance3>, check the status of the 3 supervised entities using their Health channels.	-Health status of the supervised entity 1 is kOK -Health status of the supervised entity 2 is kFailed -Health status of the supervised entity 3 is kExpired
Step 9	Waiting for the configured time <Time between qualification of SE expiry and the recovery action>	-Process of [PHMApp03] resets

16.2.2 [STS_HM_00002] HM for Deadline Supervision

Test Objective	Verification that the PHM management functional cluster can perform Deadline Supervision and do the configured recovery actions		
ID	STS_HM_00002	**State**	Draft
Affected Functional Cluster	PHM		
Trace to RS Criteria	[RS_HM_09235]		
Reference to Test Environment	STC_PHM_00001 in Test configurations of Health Monitoring		

▽

△

Configuration Parameters	- Configuring (per application), for PHMApp01, PHMApp02, PHMApp03:Configured time of Deadline Supervision Parameters Source to Target checkpoints -Supervised entity1 of MinDeadline1, MaxDeadline1 -Supervised entity2 of MinDeadline2, MaxDeadline2 -Supervised entity3 of MinDeadline3, MaxDeadline3 -MinMargin1, MinMargin2, MinMargin3 -ExpiredTolerance1, ExpiredTolerance2, ExpiredTolerance3 -ApplicationRecoveryAction is <Reset the Process> - Configured Manifest with Platform Health management - Machine state shall be Driving, in which all System Test Applications shall start.	
Summary	-Health Monitoring shall examine the Deadline supervision of 3 supervised entities. They shall report their checkpoints at the proper timing, within the configured margins. -Then after enough time, application [PHMApp02] shall miss some checkpoints reporting, yet it was for a short time, that the supervised entity went to failed state but not expired. -Then after another enough time, the application [PHMApp 03] shall miss some checkpoints reporting, to the extent that the supervised entity went to expired, leading to p rocess reset.	
Pre-conditions	- PHM Tester is connected to ECU via TCP. - Software components on ECU are initialized. - ECU is in Machine State Startup and Operating system on ECU has booted.	
Post-conditions	TCP connection between PHM Tester and ECU is closed.	
Main Test Execution		
Test Steps		**Pass Criteria**
Step 1	[PHMApp01], [PHMApp02], [PHMApp03] All applications are reporting their Transition (Source to Target) checkpoints with the correct timing more than <MinDeadline> and less than <MaxDeadline>.	
Step 2	[PHMAppCheck] After some enough time to Check the checkpoints status(S ource to Target) of 3 supervised entities using their Health channels.	-Health status of the 3 supervised entities is kOK
Step 3	[PHMApp02] Supervised Entity2 missing some of its Source to Target checkpoints, yet with in its configured Deadline time <MinDeadline1>, <MaxDeadline1>	
Step 4	[PHMAppCheck] After time corresponding Check the checkpoints status(Source to Target) of 1 supervised entity using their Health channels.	-Health status of the supervised entity 1 is kOK
Step 5	[PHMApp02] Supervised Entity 2 is reporting its Target checkpoints with exceeding configured time <MaxDeadline2>, yet for t he time below than <ExpiredTolerance2>	
Step 6	[PHMAppCheck] After some time, corresponding <ExpiredTolerance2> Check the checkpoints status (Target) of 2 supervised entity using their Health channels.	-Health status of the supervised entities 2 is kFailed.

▽

△

Step 7	[PHMApp03]. Supervised Entity 3 is reporting its Target checkpoints with time less than <MinDeadline3>, and for time more than <ExpiredTolerance3>.	
Step 8	[PHMAppCheck]. After some time, corresponding <ExpiredTolerance3> to Check the checkpoints status (Source to Target) of 3 supervised entity using their Health channels	-Health status of the supervised entity 1 is kOK -Health status of the supervised entity 2 is kFailed -Health status of the supervised entity 3 is kExpired
Step 9	Waiting for the configured time <Time between qualification of SE expiry and the recovery action>	-Process of [PHMApp03] resets

16.2.3 [STS_HM_00003] HM for Logical Supervision

Test Objective	Verification that the PHM management functional cluster can perform Logical Supervision and do the configured recovery actions		
ID	STS_HM_00003	**State**	Draft
Affected Functional Cluster	PHM		
Trace to RS Criteria	[RS_HM_09222]		
Reference to Test Environment	STC_PHM_00001 in Test configurations of Health Monitoring		
Configuration Parameters	-Configured Graph of Checkpoints(CP) (initial and Final) Connected by transitions. -Configuring (per application), for PHMApp01, PHMApp02: configured Graph of 1 to 1 Sequential process of Logical Supervision CheckPoint(CP) task (A, B, C), Here A, B, C is program task of CP -Supervised entity1 of correct sequence CheckPoint(CP) t ask (from A to B then C) -Supervised entity2 of incorrect sequence CheckPoint(CP) task (from A to C then B) -ExpiredTolerance1, ExpiredTolerance2 -ApplicationRecoveryAction is <Reset the Process> - Configured Manifest with Platform Health management - Machine state shall be Driving, in which all System Test Applications shall start.		
Summary	-Health Monitoring shall examine the Logical supervision of supervised entities. They shall report their logical checkpoints execute with configured correct sequence and incorrect sequence. -[PHMApp01] are reporting their correct sequence of Logical CP Transition (Source to Target) checkpoints and fin ally health status of Supervised1 is executed. -Then after enough time, application [PHMApp02] shall miss some checkpoints reporting, yet it was for a short time, that the supervised entity went to failed state but not expired. -Then after another enough time, the application [PHMApp 02] shall miss some checkpoints reporting, to the extent that the supervised entity went to expired, leading to process reset.		
Pre-conditions	- PHM Tester is connected to ECU via TCP. - Software components on ECU are initialized. - ECU is in Machine State Startup and Operating system on ECU has booted.		
Post-conditions	TCP connection between PHM Tester and ECU is closed.		
Main Test Execution			
Test Steps		**Pass Criteria**	

▽

△

Step 1	[PHMApp01], [PHMApp02] All applications are reporting their correct sequence of Logical CP Transition (Source to Target) checkpoints within the Graph (initial and Final).	
Step 2	[PHMAppCheck] After some enough time to Check the logical checkpoints status(Source to Target) of 2 supervised entities using their Health channels.	-Health status of the 2 supervised entities is kOK
Step 3	[PHMApp01] Supervised Entity1 of Logical checkpoints execute correct sequence as per configured CP task (from A to B then C)	
Step 4	[PHMAppCheck] After time corresponding Check the Logical checkpoints status(Source to Target) of 1 supervised entity using their Health channels.	-Health status of the supervised entity 1 is kOK
Step 5	[PHMApp02] Supervised Entity 2 of Logical checkpoints execute incorrect sequence as per configured CP task (from A to C then B), yet for time less than <ExpiredTolerance2>	
Step 6	[PHMAppCheck] After some time, to Check the Logical checkpoints status of 2 supervised entity using their Health channels.	-Health status of the supervised entities 2 is kFailed.
Step 7	[PHMApp02]. Supervised Entity 2 is missing some Logical checkpoints execute correct sequence as per configured CP task (from A to B then C), and for time more than <ExpiredTolerance 2>.	
Step 8	[PHMAppCheck]. After some time, corresponding to <ExpiredTolerance2> Check the Logical checkpoints status of 2 supervised entity using their Health channels.	-Health status of the supervised entity 2 is kExpired
Step 9	Waiting for the configured time <Time between qualification of SE expiry and the recovery action>	-Process of [PHMApp03] resets

16.2.4 [STS_PHM_00004]Determination of Local Supervision Status from Supervised Entity.

Test Objective	Verification, that the PHM management functional cluster can perform a Local Supervision Status of PHM App		
ID	STS_PHM_00001	**State**	Draft
Affected Functional Cluster	Platform Health Monitoring		
Trace to RS Criteria	[RS_PHM_00111],		

▽

△

Reference to Test Environment	STC_PHM_00001 in Test configurations of Health Monitoring	
Configuration Parameters	- Configuring (per application), for PHMApp01, PHMApp02, PHMApp03. - SupervisionCycle1, SupervisionCycle2, SupervisionCycle3. -FailedSupervisionCyclesTolerance1, FailedSupervisionCyclesTolerance2, FailedSupervisionCyclesTolerance3. -ExpiredSupervisionCyclesTolerance1, ExpiredSupervisionCyclesTolerance2, ExpiredSupervisionCyclesTolerance3. -ApplicationRecoveryAction is <Reset the Process>. -Health Monitoring Contribution to Machine. -Machine State Driving, in which all System Test Applications [App01] shall start is defined.	
Summary	- Health Monitoring Initial Supervision Mode (Initial Mode) (i.e. each Supervised Entity that is activated in the initial mode). then to verify all possible state of local Supervision status of Supervised Entity.	
Pre-conditions	- PHM Tester is connected to ECU via TCP. - Software components on ECU are initialized. - ECU is in Machine State Startup. -Operating system on ECU has booted.	
Post-conditions	- TCP connection between PHM Tester and [ECU] is closed.	
Main Test Execution		
Test Steps		**Pass Criteria**
Step 1	[PHMApp01], [PHMApp02], [PHMApp03] All applications are reporting their Local Supervision Status of a Supervised Entity result of Alive Supervision, result of Deadline Supervision, result of Logical Supervision is executed within <SupervisionCycle>.	
Step 2	[PHMAppCheck] Get the Local Supervision status of [PHMApp01], [PHMApp02], [PHMApp03] of Supervised Entity.	-Supervised Entity of LOCAL STATUS OK
Step 3	[PHMApp01] Report incorrect result of Alive Supervision with configured and for time more than <ExpiredSupervisionCyclesTolerance1> < FailedSupervisionCyclesTolerance1=0>, and incorrect result of Deadline or Logical supervision of Supervised Entity.	
Step 4	[PHMAppCheck] After time corresponding to 100 x the longest < SupervisionCycle1 >, Get the Local Supervision status of [PHMApp01].	-Supervised Entity of LOCAL STATUS EXPIRED
Step 5	[PHMApp01] Report incorrect Alive Supervision with configured < FailedSupervisionCyclesTolerance1 =1 >, and correct Deadline, Logical supervision of Supervised Entity.	
Step 6	[PHMAppCheck] After time corresponding to 100 x the longest < SupervisionCycle1 >, Get the Local Supervision status of [PHMApp01].	-State change to LOCAL STATUS FAILED
Step 7	[PHMApp01] Report correct Alive Supervision with configured < FailedSupervisionCyclesTolerance1 >1 >, and correct Deadline, Logical supervision of Supervised Entity.	

▽

△

Step 8	[PHMAppCheck] After time corresponding to 100 x the longest < SupervisionCycle1 >, Get the Local Supervision status of [PHMApp01].	-State change to LOCAL STATUS FAILED
Step 9	[PHMApp01] Report correct Alive Supervision with configured FailedSupervisionCyclesTolerance1 =1, and correct Deadline, Logical supervision of Supervised Entity.	
Step 10	[PHMAppCheck] After time corresponding to 100 x the longest < SupervisionCycle1 >, Get the Local Supervision status of [PHMApp01].	-State change to LOCAL STATUS OK
Step 11	Health monitoring is switch to the mode and Change status	LOCAL STATUS DEACTIVATED.

16.2.5 [STS_PHM_00005] Determination of Global Supervision Status from Supervised Entity.

Test Objective	Verification, that the PHM management functional cluster can perform a global Supervision Status of PHM App.		
ID	STS_PHM_00005	**State**	Draft
Affected Functional Cluster	Platform Health Monitoring		
Trace to RS Criteria	[RS_PHM_00111]		
Reference to Test Environment	STC_PHM_00001 in Test configurations of Health Monitoring		
Configuration Parameters	- Configuring (per application), for PHMApp01, PHMApp02, PHMApp03. - SupervisionCycle1, SupervisionCycle2, SupervisionCycle3. -FailedSupervisionCyclesTolerance1, FailedSupervisionCyclesTolerance2, FailedSupervisionCyclesTolerance3. -ExpiredSupervisionCyclesTolerance1, ExpiredSupervisionCyclesTolerance2, ExpiredSupervisionCyclesTolerance3. -ApplicationRecoveryAction is <Reset the Process>. -Health Monitoring Contribution to Machine. -Machine State Driving, in which all System Test Applications [App01] shall start is defined.		
Summary	- Health Monitoring Initial Supervision Mode (Initial Mode) (i.e. each Supervised Entity that is activated in the initial mode). then to verify all possible state of Global Supervision status of Supervised Entity.		
Pre-conditions	- PHM Tester is connected to ECU via TCP. - Software components on ECU are initialized. - ECU is in Machine State Startup. -Operating system on ECU has booted.		
Post-conditions	-TCP connection between PHM Tester and [ECU] is closed.		
Main Test Execution			
Test Steps			**Pass Criteria**

▽

△

Step 1	[PHMApp01], [PHMApp02], [PHMApp03] All applications are reporting their Local Supervision Status of a Supervised Entity result of Alive Supervision, result of Deadline Supervision, result of Logical Supervision is executed within <SupervisionCycle>.	
Step 2	[PHMAppCheck] Get the Local Supervision status of [PHMApp01], [PHMApp02], [PHMApp03] of Supervised Entity.	-Supervised Entity of LOCAL STATUS OK
Step 3	[PHMApp01] Report incorrect result of Alive Supervision with configured and for time more than <ExpiredSupervisionCyclesTolerance1> < FailedSupervisionCyclesTolerance1=0>, and incorrect result of Deadline or Logical supervision of Supervised Entity.	
Step 4	[PHMAppCheck] After time corresponding to 100 x the longest < SupervisionCycle1 >, Get the Local Supervision status of [PHMApp01].	-Supervised Entity of LOCAL STATUS EXPIRED
Step 5	[PHMApp01] Report incorrect Alive Supervision with configured < FailedSupervisionCyclesTolerance1 =1 >, and correct Deadline, Logical supervision of Supervised Entity.	
Step 6	[PHMAppCheck] After time corresponding to 100 x the longest < SupervisionCycle1 >, Get the Local Supervision status of [PHMApp01].	-State change to LOCAL STATUS FAILED
Step 7	[PHMApp01] Report correct Alive Supervision with configured < FailedSupervisionCyclesTolerance1 >1 >, and correct Deadline, Logical supervision of Supervised Entity.	
Step 8	[PHMAppCheck] After time corresponding to 100 x the longest < SupervisionCycle1 >, Get the Local Supervision status of [PHMApp01].	-State change to LOCAL STATUS FAILED
Step 9	[PHMApp01] Report correct Alive Supervision with configured FailedSupervisionCyclesTolerance1 =1, and correct Deadline, Logical supervision of Supervised Entity.	
Step 10	[PHMAppCheck] After time corresponding to 100 x the longest < SupervisionCycle1 >, Get the Local Supervision status of [PHMApp01].	
Step 11	[PHMAppCheck] [PHMApp01], [PHMApp02] applications are reporting their Local Supervision Status of a Supervised Entity result.	State change to LOCAL STATUS OK
Step 12	[PHMAppCheck] [PHMApp01] Report the Supervised Entity Instance is LOCAL STATUS OK, no Failed instance.	[PHMApp01] GLOBAL SUPERVISON STATUS OK

▽

△

Step 13	[PHMAppCheck] After time corresponding to 100 x the longest < SupervisionCycle2 >, Get the Local Supervision status of [PHMApp02], Supervised Entity Instance is LOCAL STATUS FAILED and no Supervised Entity Instance is in Local Supervision Status LOCAL STATUS EXPIRED.	-Supervision status of [PHMApp02],GLOBAL STATUS FAILED.
Step 14	[PHMAppCheck] After time corresponding to 100 x the longest < SupervisionCycle3 >, Get the Local Supervision status of [PHMApp03], Instance is LOCAL STATUS EXPIRED and the expired <ExpiredSupervisionCyclesTolerance3>is configured to a value larger than zero.	-Supervision status of [PHMApp03],GLOBAL STATUS EXPIRED
Step 15	[PHMAppCheck] After time corresponding to 100 x the longest < SupervisionCycle3 >, Get the Local Supervision status of [PHMApp03], Supervised Entity Instance is LOCAL STATUS EXPIRED and the ExpiredSupervisionCyclesTolerance3=0 is configured to zero.	-[PHMApp03],Supervision status of GLOBAL STATUS STOPPED.

17 Test configuration and test steps for State Management

17.1 Test System

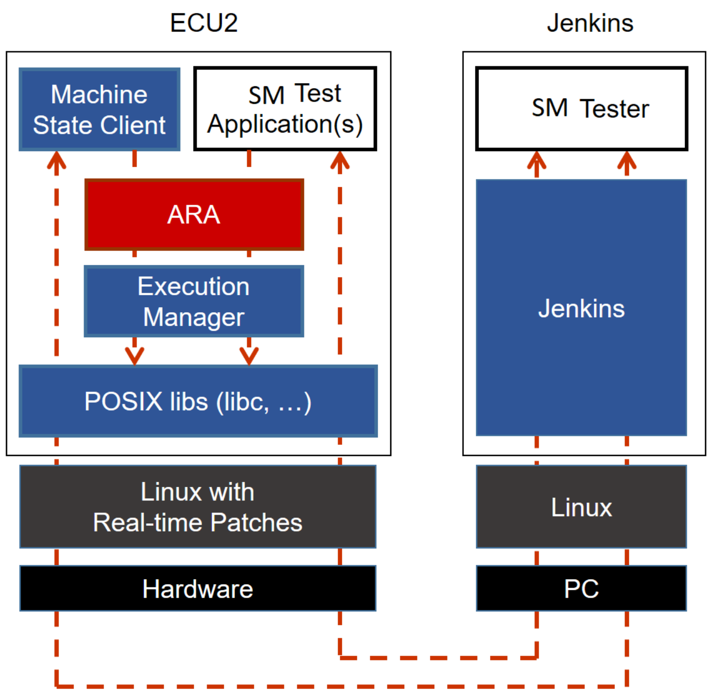

Figure 17.1: Illustration of test setup for State Management.

17.1.1 Test configurations

17.1.1.1 STC_SM_00001

Configuration ID	STC_SM_00001
Description	Standard Jenkins server for State Management test
ECU 2	Hardware, 192.168.100.2
Jenkins	Jenkins Server, 192.168.100.10

The Jenkins Server, running the job with the State Management test (SM Tester) is connected via Ethernet to ECU2 hosting the System Test Applications [SMApp02], [SMApp03], [SMApp04], [SMApp05] and [SMApp06].

The Exec Tester is supposed to check the pass criteria.

The communication between Exec Tester and the applications on the ECU may take place over the Diagnostics functional cluster in form of diagnostic messages.

17.1.1.1.1 Machine Manifest

Machine States	*Startup (Initial Mode)*
	Shutdown
	Restart
	Driving
	Parking
Function Groups	
FG1	*Off*
	State1
	State2
	State3

17.1.1.1.2 Application Manifest

Application Name	**SMApp02**		
Modelled Process	ModeDependentStartupConfig	machineMode	*StartUp*
Application Name	**SMApp03**		
Modelled Process	ModeDependentStartupConfig	functionGroup	*StartUp*
Application Name	**SMApp04**		
Modelled Process1	ModeDependentStartupConfig	machineMode	*StartUp*
Modelled Process2	ModeDependentStartupConfig	functionGroup	*State1*
Application Name	**SMApp05**		
Modelled Process	ModeDependentStartupConfig	functionGroup	*State2*
Application Name	**SMApp06**		
Modelled Process	ModeDependentStartupConfig	functionGroup	*State3*

17.2 Test cases

17.2.1 [STS_SM_00001] Evaluate State Management shall coordinate and control multiple sets of Applications.

Test Objective	Verification that the State Management shall coordinate and control multiple sets of Applications.		
ID	STS_SM_00001	**State**	Draft
Affected Functional Cluster	State Management		
Trace to RS Criteria	[RS_SM_00001]		

▽

△

Reference to Test Environment	STC_SM_00001
Configuration Parameters	- Service Interface - *TriggerIn_StateGroup1* - Method - *SM_RequestState_Int1* - Service Interface - *TriggerOut_StateGroup1* - Method - *SM_StateChangeEvent_Int1* - SMApp02 Modelled Process, SMApp03 Modelled Process and SMApp04 Modelled Process1 are configured to be started in Machine State StartUp - SMApp04 Modelled Process2 is configured to be started in Function Group FG1 state *State1* - SMApp05 Modelled Process is configured to be started in Function Group FG1 state *State2* - SMApp06 Modelled Process is configured to be started in Function Group FG1 state *State3* - SMApp02 Modelled Process is configured with Trigger Field *TriggerIn_StateGroup1* - SMApp03 Modelled Process is configured with Trigger Field *TriggerIn_StateGroup1* - SMApp04 Modelled Process1 is configured with Notifier Field *TriggerOut_StateGroup1* - ECU ID for ECU2 is set to *ECU2* - SMApp05 Application has LT Application ID *APPID5* - Context Id for SMApp05 Application is set to *CTX5*

▽

△

Summary	Internal states of a state machine *SM_Int1* in State Management is associated to Function Group FG1 SM_Int_Off : Off (FG1 state) SM_Int_State1 : State1 (FG1 state) SM_Int_State2 : State2 (FG1 state) SM_Int_State3 : State3 (FG1 state) Whenever State Management changes to above internal states it shall request the state changes as mentioned above to Execution Manager w.r.t Function Group FG1 State Management handling is project specific and following assumptions are made in the test for handling the states SM_Int_Off, SM_Int_State1, SM_Int_State2 and SM_Int_State3 states which is associated to Function Group FG1 Whenever a request is ongoing state Management shall queue the requests based on FIFO method The state machine *SM_Int1* is currently in state *SM_Int_State1* and Function Group FG1 is in state *State1* SMApp02 application Modelled Process requests for a state change to *SM_Int_State2* state via method call *SM_RequestState_Int1* using ara::com. The State Manager checks internally and decides to process this request and requests for a state change to *State2* to Execution Manager via API SetState(FunctionGroupState state) Before the Execution manager shall respond to the state change *State2*. SMApp03 application Modelled Process requests for a state change to *SM_Int_State3* state via *SM_RequestState_Int1* using ara::com. The State Manager shall queue the request from SMApp03 application Modelled Process. Execution manager shall stop SMAPP04 application Modelled Process2 and then start the SMApp05 application Modelled Process and then returns void as a return for SetState API to indicate that the requested transition was successful. State Management changes its internal state to *SM_Int_State2* and shall trigger event *SM_StateChangeEvent_Int1* to notify SMApp04 Modelled Process1 application by invoking ara::com api send(&state) to inform about the new state *State2*. On receiving the new state, SMApp04 Modelled Process2 invokes an internal function SMApp04Func within which the DLT log message with MSG1 with new state *State2* is reported. State Manager shall then process the request from SMApp03 application Modelled Process and requests for a state change to *State3* to Execution Manager via API SetState(FunctionGroupState &state) Execution manager shall stop SMAPP05 application Modelled Process and then start the SMApp06 application Modelled Process and then returns void as a return for SetState API to indicate that the requested transition was successful. State Management changes its internal state to *SM_Int_State3* and shall trigger event *SM_StateChangeEvent_Int1* to notify SMApp04 Modelled Process1 application by invoking ara::com api send(&state) to inform about the new state *State3*.	
Pre-conditions	- SM Tester is connected to ECU2 via TCP. - Software components on ECU2 are initialized. - ECU2 is in Machine State *Startup*. - ECU2 is in Function Group FG1 State *State1*. - Operating system on ECU2 has booted.	
Post-conditions	TCP connection between Exec Tester and ECU2 is closed.	
Main Test Execution		
Test Steps		**Pass Criteria**
Step 1	[SM Tester] Query execution status of [SMAPP04] Modelled Process2.	[SMAPP04] Modelled Process2 is executed
Step 2	[SMApp02] Request change of Internal State to *SM_Int_State2* from State Manager via method call *SM_RequestState_Int1*	
Step 3	[SMApp03] Request change of Internal State to *SM_Int_State3* from State Manager via method call *SM_RequestState_Int1*	

▽

△

Step 4	[SM Tester] Request for change of Function Group FG1 State to *State2* from Execution Manager by invoking SetState API.	
Step 5	[SM Tester] Query execution status of [SMAPP04] Modelled Process2.	[SMAPP04] Modelled Process2 is not executed.
Step 6	[SM Tester] Query execution status of [SMAPP05] Modelled Process.	[SMAPP05] Modelled Process is executed.
Step 7	[SM Tester] Observe the log for [SMAPP04] Modelled Process2.	Message with MSG1 new state *State2* is received Message with context ID *CTX42* and application ID *APPID4* is received which is logged within the internal function *SMApp04Func* of [SMAPP04] Modelled Process2
Step 8	[SM Tester] Request for change of Function Group FG1 State to *State3* from Execution Manager by invoking SetState API.	
Step 9	[SM Tester] Query execution status of [SMAPP05] Modelled Process.	[SMAPP04] Modelled Process2 is not executed.
Step 10	[SM Tester] Query execution status of [SMAPP06] Modelled Process.	[SMAPP06] Modelled Process is executed .
Step 11	[SM Tester] Observe the log for [SMAPP04] Modelled Process2.	Message with MSG2 new state *State3* is received. Message with context ID *CTX42* and application ID *APPID4* is received which is logged within the internal function *SMApp04Func* of [SMAPP04] Modelled Process2

18 References

[1] Glossary AUTOSAR_TR_Glossary