Bundesamt
für Sicherheit in der
Informationstechnik

BSI Technical Guideline 03125
Preservation of Evidence of Cryptographically Signed Documents

Annex TR-ESOR-C.1:
Conformity Test Specification (Level 1 - Functional Conformity)

Designation	Functional Conformity Test Specification (Level 1)
Abbreviation	BSI TR-ESOR-C.1
Version	1.2
Date	19.02.15

Part of the document is not barrier free.

Federal Office for Information Security
Post Box 20 03 63
53133 Bonn
Phone: +49 228 99 9582-0
E-Mail: tresor@bsi.bund.de
Internet: https://www.bsi.bund.de

Table of Contents

Illustration Index

1. Introduction

The goal of the Technical Guideline “Preservation of Evidence of Cryptographically Signed Documents” is to specify technical security requirements for the long-term preservation of evidence of cryptographically signed electronic documents and data along with associated electronic administrative data (meta data).

A Middleware defined for this purpose (TR-ESOR-Middleware) in the sense of this Guideline includes all of the modules (**M**) and interfaces (**S**) [for the German "*Schnittstellen*"] used for securing and preserving the authenticity and proving the integrity of the stored documents and data.

The Reference Architecture introduced in the Main Document of this Technical Guideline consists of the functions and logical units described in the following:

- The input interface S.4 of the TR-ESOR-Middleware serves to embed the TR-ESOR-Middleware in the existing IT and infrastructure landscape;
- The central Middleware module (**[TR-ESOR-M.1]**), which regulates the flow of information in the Middleware, that implements the security requirements for the interfaces with the IT applications and which ensures that the application systems are decoupled from the ECM/long-term storage;
- The “Cryptographic" module (**[TR-ESOR-M.2]**) and the associated interfaces S.1 and S.3 that provide the functions needed for the creation (optional) and verification of electronic signatures, the post-verification of electronic certificates, and for the obtainment of qualified time stamps for the Middleware. Furthermore, it can provide the functions for the encryption and decryption of data and documents;
- The “ArchiSig” module (**[TR-ESOR-M.3]**) with the interface S.6 that provides the functions needed for the preservation of evidence of the digitally signed documents;
- An ECM/long-term storage with the interfaces S.2 and S.5 that assumes the physical archiving/storage and also the storage of the meta data that preserve evidence.
  This ECM/long-term storage is no longer directly a part of the Technical Guideline, but requirements may be induced through the two interfaces that are still part of the TR-ESOR-Middleware.
  The application layer that can include an XML-adapter is not a direct part of this Technical Guideline, either, even though this XML-adapter can be implemented as part of a Middleware.

The IT Reference Architecture depicted in Figure 1 is based on the ArchiSafe[1] Reference Architecture [PTB 05] and is supposed to make possible and support the logical (functional) interoperability of future products with the goals and requirements of the Technical Guideline.

[1] For more information, see http://www.archisafe.de.

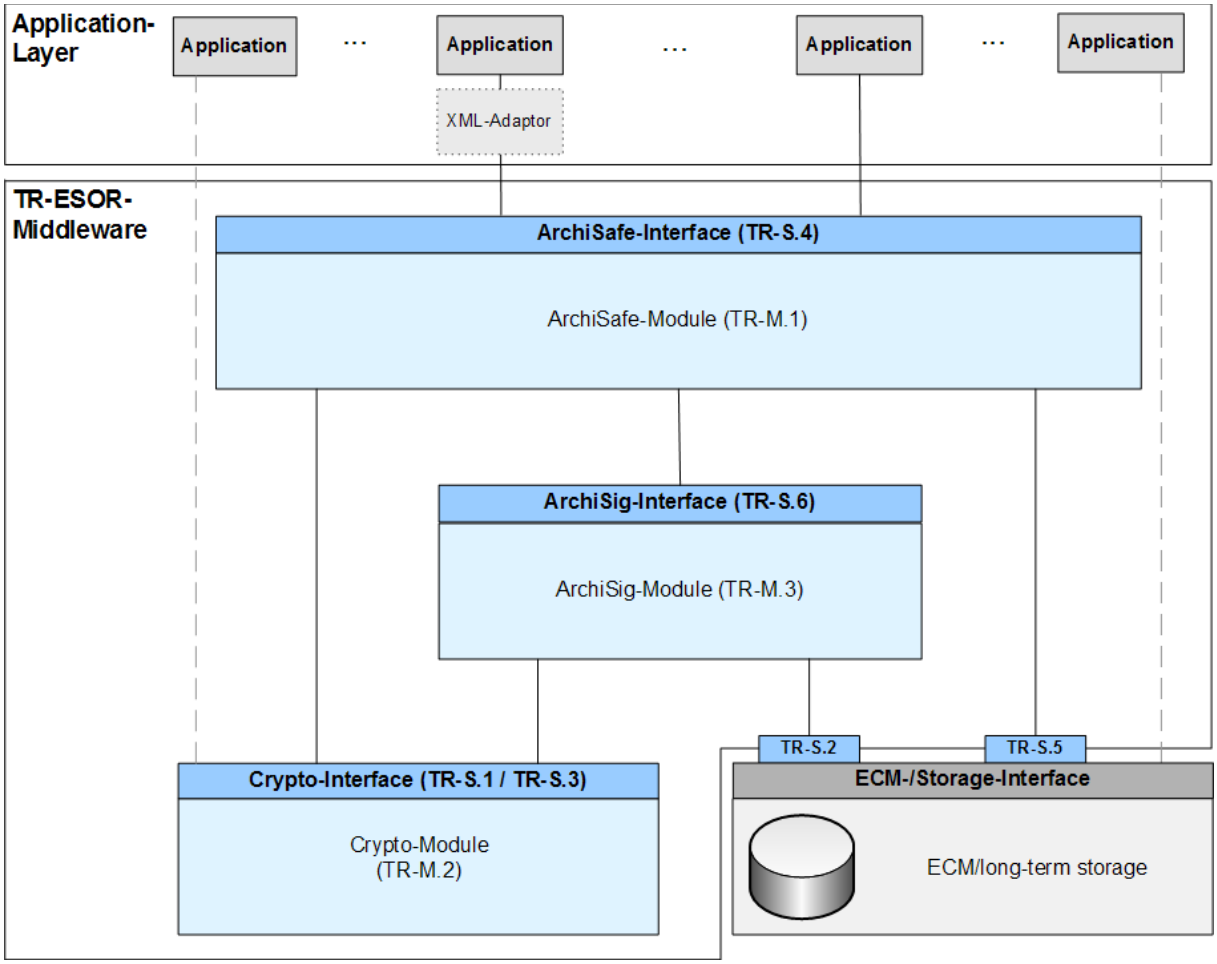

Figure 1: Schematic Depiction of the IT Reference Architecture

This Technical Guideline is modularly structured, and the individual annexes to the Main Document specify the functional and technological security requirements for the needed IT components and interfaces of the TR-ESOR-Middleware. The specifications are strictly platform, product, and manufacturer independent.

The document at hand bears the designation “Annex TR-ESOR-C.1” and describes and specifies the conformity tests for the conformity level 1 “Functional Conformity”.

2. Overview

Products or systems which want to get certified according to this Technical Guideline have to demonstrate their conformance to the specifications. There are three conformance levels defined which mainly differ in the technical detail specifications of interfaces and data formats used.

- Conformity Level 1 – Functional Conformity
- Conformity Level 2 – Technical Conformity
- Conformity Level 3 – Recommendations for Federal Agencies

The three levels are built on top of each other. This means e.g. in order to demonstrate conformity to level 2 all conformance criteria for level 1 have to be passed in addition to the conformance criteria for level 2.

This document specifies the functional conformity criteria (tests) derived from the requirements specified in the documents of the Technical Guideline.

In order to become certified according to a conformity level, a product or system must pass all conformity criteria (tests) for this conformity level and for all lower conformity levels. If one or more tests are not successful, the conformity cannot be certified.

In the following chapter the test criteria will be derived from the requirements defined of the TR. Furthermore, the requirements and therefore also the test criteria are assigned to a conformity level.

Based on these assignments the subsequent chapters define the test cases for the conformity levels in detail. Red headlined (marked) test cases MUST be passed for fulfilling the conformity criteria.

The test case specifications are written in such a way that this document (or the respective parts of it) could be used as template for the documentation of the final results of actual testing.

3. Test Approach

The following test specifications are based on the recommended reference architecture in chapter 7.1 of the main document of this technical guideline. Thus, in the following differences between expected and observed test results should be carefully interpreted by the testers respecting the fact that actual implementations of components and / or modules of the middleware may deviate from the recommended reference architecture. This may result also in different characteristics of implemented and provided interfaces.

Beside this testing the conformity to this guideline may refer to a single module only. This may result also in different characteristics and expected results of implemented and provided features and interfaces.

In the following text we use the wording "S.4 Interface" instead of "S.4 Interface or functionally analogous interfaces". It is worth noting, therefore, that testing the conformity level 1 the referred interfaces are required in a logical functional manner only and not in a technical interoperable characteristic.

The TR-ESOR interfaces S.2 and S.5 are actually not part of the TR-ESOR middleware because they will be provided by the storage system. Therefore, no conformity tests will be specified here.

For fulfilling the required conformity in general, the red marked test specifications in this document must be tested and passed.

For fulfilling the required conformity in compliance with the pre-suppositions written down before the test cases, the yellow marked test specifications in this document must be tested and passed.

All other test specifications must be passed or the non-fulfilment must be justified.

3.1 Structure of the Test Case Specifications

Some test cases are ordered according to the modules M.1 – M.3 and „all products". These test cases cannot be assigned to the certain interface of the module but check general properties of the module.

The other test cases are ordered according to the interface specifications S.1 – S.6. The reason for that is that these tests will only be performed on the level of external interfaces of a certain product. If a product claims compliance with the module specified in the Technical Guideline, the respective interfaces of the module (product) will be tested or the product proves that it supports functional analogous interfaces.

Below this structural level, the test cases are ordered according to the logical functions of this interface, e.g. „Archive Submission" or „Archive Deletion". For each logical function of the interface a set of test cases test all relevant requirements.

Each test case is identified by a unique ID. The test case description also refers to the respective requirements which will be (partly) tested with this test case. The test case also states the purpose of the test as a summary of the test case. The baseline configuration of the test system will be stated as well as all pre-conditions which must exist prior performance of the test. The test case defines the single test steps which must be performed in the given order. Per test step the expected result is defined and there is space that the tester could document the actual findings. Finally, the tester can state the final verdict of the test case (PASS/FAIL).

FAIL shall be assigned if any of the test steps does not match the expected result and a justification for this difference is not possible.

3.2 Strictness of Test Result Assessment

The Technical Guideline differs between three major classes of requirements (cf. [RFC 2119])

- CAN (or synonymously MAY, COULD) : These requirements are just hints or optional features. These requirements will not be tested.
- SHOULD: These requirements are strong recommendations. Respective test cases should

demonstrate the specified behaviour. Alternatively, the vendor explains why its product uses another approach and why the resulting security level is equal to the security level described in the Technical Guideline.

- MUST (or synonymously SHALL): These are strict requirements. It is not allowed to use another approach or alternative techniques.

Test cases which tests MUST requirements are identified with a red coloured title line. The expected results of these test cases must exactly be the actual results.

Test cases identified by a grey coloured title line are pure SHOULD requirements. The expected test results may differ from the actual test results, if the vendor can demonstrate the same or higher security level.

3.3 Baseline for all Test Cases

This section describes the basics valid and usable for all test cases.

3.3.1 Standard Test Configurations

Here, a set of standard configurations of the test setup will be described. These setups are referenced in the test cases and should be used to actually perform the tests.

3.3.1.1 CONFIG_Common

This is the standard configuration for all tests.

- The test setup shall contain the product to be tested (Target of Testing, TOT).
- The test setup shall contain all other modules of the reference architecture (including the storage) functionally not covered by the TOT.
  The purpose is that a functionally complete system can be tested.
- The TOT and all other modules required shall be installed and configured according to the respective guidance including all security recommendations.
- The TOT and all other modules shall be physically and logically interconnected. The connections shall be secured as described in the respective guidance documents (e.g. enabling encryption, explicit physical connection).
- The test system shall be connected to an external Certification Service Provider as required by the TOT or the tests.
- At least it is recommended to install three different client applications for using and testing the multi-client-capability of the middleware (if the TOT supports/provides a multi-client-capability).
  In this case the middleware in turn shall be configured to handle these three applications as different clients (multi-client-capability). Per client application at least two user accounts and an administrator account shall be configured.

The complete test setup shall be up and running and in an operational and working mode.

3.3.1.2 CONFIG_ArchiSafe

This configuration is based on CONFIG_Common.

Additionally, the ArchiSafe-Module (if TOT) shall be configured as follows:

- If configurable, a XSD defining the XAIPs shall be configured. Preferable, the XAIP described in Annex TR-ESOR-F should be used.
- If configurable, the XSD verification of XAIP containers during Archive Submission and Archive Update shall be enabled.
- If configurable, the signature verification[2] during Archive Submission and Archive Update shall be enabled.
- If configurable, the S.4 interface shall only be accessible using a secure Channel (e.g. TLS

[2] The verification of signatures of documents included in the XAIP or passed over as binary.

tunnel) with certificate-based mutual authentication.

3.3.2 Standard Test Objects

For most of the tests test data is required. In order to make the tests repeatable, this section defines some standard test objects.

The following test objects are available for the three Conformity Levels 1, 2 and 3.

Table 1: Definition of test objects

No.	Container Name	Used in Conformity Level	Description
1	XAIP_OK	1, 2, 3	The XAIP is syntactically correct and passes the defined consistency checks.
2	XAIP_OK_SIG	1	The XAIP is syntactically correct and passes the defined consistency checks and there is a valid signature.
3	XAIP_NOK	1, 2, 3	The schema validation of the XAIP fails.
4	XAIP_NOK_EXPIRED	1,2	The schema validation for the XAIP succeeds, but the preservationInfo-element indicates a preservation date, which is already exceeded.
5	XAIP_NOK_SUBMTIME	2	The schema validation for the XAIP succeeds, but the submissionTime-element deviates from the current time beyond a reasonable tolerance range. The documentation of the middleware or the module, which shall be tested, **shall** contain some assertions and related conditions or constraints indicating when the submissionTime contained in the provided XAIP deviates too much from the current time.
6	XAIP_NOK_SIG	1, 2,	The XAIP is syntactically correct and passes the defined consistency checks, but the XAIP contains an invalid signature. Invalid signature means that the signature is syntactically not correct or at least one of the evidence relevant data, for example a signature or timestamp or certificate or revocation list or OCSP-response, etc., is wrong.
7	XAIP_NOK_ER	1, 2	The XAIP is syntactically correct and passes the defined consistency checks, but the XAIP contains an invalid Evidence Record. Invalid Evidence Record means, that the Evidence Record is syntactically not correct or does not pass the defined consistency checks according to annex C.2, chapter 4.1. or annex ERS.
8	XAIP_NOK_SIG_OK_ER	1, 2	The XAIP is syntactically correct and passes

No.	Container Name	Used in Conformity Level	Description
			the defined consistency checks and there is a correct Evidence Record or a number of correct Evidence Records, but the XAIP contains a signature, which was not correct at the time of its archiving.
9	XAIP_OK_SIG_OK_ER	1, 2	The XAIP is syntactically correct and passes the defined consistency checks and there is a valid signature and a valid Evidence Record or a number of correct Evidence Records.
10	BIN	1, 2, 3	This test object is a binary document, which is provided in the ArchiveData-element.
11	XAIP(BIN)	1, 2	The XAIP(BIN) is a XAIP, which is part of the response of a successful ArchiveRetrievalRequest concerning an archive data object, which was previously inserted as a BIN in the long-term storage by an ArchiveSubmissionRequest.
12	DXAIP_OK	1, 2, 3	The DXAIP is syntactically correct and represents a valid update container (“Delta XAIP”) for XAIP_OK, which contains the corresponding AOID.
13	DXAIP_OK_SIG	1	The DXAIP is syntactically correct and represents a valid update container (“Delta XAIP”) for XAIP_OK, which contains the corresponding AOID and contains a valid signature.
14	DXAIP_NOK	1, 2, 3	The DXAIP is syntactically not correct because the schema validation fails.
15	DXAIP_NOK_AOID	2	The schema validation for the Delta XAIP succeeds, but the update container (“Delta XAIP”) contains a not yet assigned AOID.
16	DXAIP_NOK_EXPIRED	2	The schema validation for the Delta XAIP succeeds, but the preservationInfo-element indicates a point in time in the past.
17	DXAIP_NOK_SUBMTIME	2	The schema validation for the Delta XAIP succeeds, but the submissionTime-element deviates from the current time beyond a reasonable tolerance range. The documentation of the middleware or the module, which shall be tested, **shall** contain some assertions and related conditions or constraints indicating when the submissionTime contained in the provided XAIP deviates too much from the current time.

No.	Container Name	Used in Conformity Level	Description
18	DXAIP_NOK_SIG	1, 2	The schema validation for the Delta XAIP succeeds, but the XAIP contains an invalid signature.
19	DXAIP_NOK_ER	2	The schema validation for the Delta XAIP succeeds, but the XAIP contains an invalid Evidence Record.
20	DXAIP_NOK_VERSION	2	The schema validation for the XAIP succeeds, but there is a syntactical collision with the original XAIP such that the schema validation for the compound XAIP fails, for example the element prevVersion in the updateSection of the DXAIP is not the latest version of this XAIP.
21	DXAIP_NOK_ID	2	The DXAIP contains no or an invalid ID.
22	TST_OK	2	The time stamp token is syntactically correct and based on a valid signature.
23	TST_OK_VALINFO	2	This time stamp token is based on TST_OK and contains the validation information, which has been collected during verification.
24	TST_NOK	2	The time stamp token is syntactically incorrect.
25	TST_NOK_SIG	2	The time stamp token is syntactically correct, but the signature does not verify correctly.
26	TST_NOK_VALINFO	2	This time stamp token is based on TST_OK and contains validation information, which has been collected during verification, but are not complete.
27	TST_BASIS_ERS_OK	2	The time stamp token is based on ([TR-ESOR-ERS], Profil BASIS_ERS) and is syntactically correct and based on a valid signature.
28	TST_BASIS_ERS_OK_VALINFO	2	This time stamp token is based on ([TR-ESOR-ERS], Profil BASIS_ERS) and contains the validation information, which has been collected during verification.
29	TST_BASIS_ERS_NOK	2	The time stamp token is based on ([TR-ESOR-ERS], Profil BASIS_ERS) and is syntactically incorrect.
30	TST_BASIS_ERS_NOK_SIG	2	The time stamp token is based on ([TR-ESOR-ERS], Profil BASIS_ERS) and is syntactically correct, but the signature does not verify correctly.

No.	Container Name	Used in Conformity Level	Description
31	TST_BASIS_ERS_NOK_VALINFO	2	This time stamp token is based on ([TR-ESOR-ERS], Profil BASIS_ERS) and contains validation information, which has been collected during verification, but are not complete.
32	ER_OK_INIT	2	The Evidence Record according [RFC4998] and [TR-ESOR-ERS]/Basic-ERS-Profile and based on XAIP_OK contains only an initial archive timestamp.
33	ER_NOK_INIT	2	The initial archive timestamp of the Evidence Record according [RFC4998] and [TR-ESOR-ERS]/Basic-ERS-Profile and based on XAIP_OK can not be validated.
34	ER_OK_CHAIN	2	The Evidence Record according to [TR-ESOR-ERS]/Basic-ERS-Profile is based on XAIP_OK and includes an archive timestamp chain according to [RFC4998].
35	ER_NOK_CHAIN	2	The Evidence Record according to [TR-ESOR-ERS]/Basic-ERS-Profile and based on XAIP_OK includes an archive timestamp chain according to [RFC4998] which can not be validated.
36	ER_OK_SEQ	2	The Evidence Record according to [TR-ESOR-ERS]/Basic-ERS-Profile is based on XAIP_OK and includes an archive timestamp sequence according to [RFC4998].
37	ER_NOK_SEQ	2	The Evidence Record according to [TR-ESOR-ERS]/Basic-ERS-Profile and based on XAIP_OK includes an archive timestamp sequence according to [RFC4998] which can not be validated.
38	XAIP_OK_XBDP	3	Tthe test objects are enriched by the newley defined data elements of [TR-ESOR-XBDP]. The XAIP is syntactically correct and passes the defined consistency checks.
39	XAIP_NOK_XBDP	3	Tthe test objects are enriched by the newley defined data elements of [TR-ESOR-XBDP]. The schema validation of the XAIP fails.

These test objects are referred in the test cases by their unique name.

The actual test objects (the files) for this annex are provided as appendix to this document.

- “Container Name” contains the unique name of the container and is identical to the file name.
- “XML Schema”
  - “valid” means that a XML-based object conforms with the specified XML

Schema.
    - “not valid” means that a XML-based object does not conform with the specified XML Schema.
    - “---” means that this is a binary object which does not claim conformance to a XML schema.
- “Binary”
    - “no” means that this is an XML object.
    - “yes” means that this is a binary (a non-XML) object.
- “Preservation Time”
    - “Future” means that the the minimum retention date is somewhere in the future (e.g. 01.01.2100).
    - “Past” means that the the minimum retention date is somewhere in the past (e.g. 01.01.2000).
- “Signature”
    - “No signature” means that the user data contained in the test object does not contain a digital signature.
    - “Valid” means that the user data contained in the test object contains a digital signature which is mathematically correct, produced with an approved algorithm and with a valid (neither expired nor revoked) certificate issued by a known and trustworthy Certificate Authority. It does not need to be a qualified signature.
    - “Not Valid” means that the user data contained in the test object contains a digital signature which is mathematically not correct but produced with an approved algorithm and with a valid (neither expired nor revoked) certificate issued by a known and trustworthy Certificate Authority. It does not need to be a qualified signature.

Table 2 Definition of test data in detail

Container Name	XML Schema	Binary	Preservation Time	Signature
XAIP_OK	Valid	No	Future	No Signature
XAIP_OK_SIG	Valid	No	Future	Valid
XAIP_NOK_EXPIRED	Valid	No	Past	undefined
XAIP_NOK	Not Valid	No	Future	undefined
XAIP_NOK_SIG	Valid	No	Future	Not Valid
XAIP_NOK_ER	Valid	No	Future	Evidence Record NOT Valid
				Signature undefined
XAIP_NOK_SIG_OK_ER	Valid	No	Future	Evidence Record Valid
				Signature Not Valid
XAIP_OK_SIG_OK_ER	Valid	No	Future	Evidence Record Valid
				Signature Valid
XAIP(BIN)	Valid	No	Future	Evidence Record Valid
				Signature Valid
BIN	---	Yes	undefined	Signature valid
BIN_NOK_SIG	---	Yes	undefined	Signature Not Valid
DXAIP_OK	Valid	No	Future	No Signature
DXAIP_OK_SIG	Valid	No	Future	Valid signature
DXAIP_NOK	Not Valid	No	Future	undefined
DXAIP_NOK_SIG	Valid	No	Future	Not Valid

3.4 Occurring Abbreviations

Abbreviation	Meaning
AES-128	Advanced Encryption Standard (128 bits)
AOID	Archive Object Identifier
ATS	Archive Time Stamp
BIN	Binary
BSI	Federal Office for Information Security
C14N	Canonical XML Version 1.0
C14N11	Canonical XML Version 1.1
C14N20	Canonical XML Version 2.0

Abbreviation	Meaning
CA	Certification Authority
CMS	Cryptographic Message Syntax
CRL	Certificate Revocation List
DES	Data Encryption Standard
DoS	Denial of Service
e.g.	for example (exempli gratia)
EC14N	Exclusive XML Canonicalization
ECM	Enterprise Content Management
ERS	Evidence Record Syntax
ETSI-TSP	European Telecommunication Standard Institut - Time Stamping Profile
HTTP	Hypertext Transfer Protocol
i.e.	in other words (id est)
ID	Identifier
IT	Information Technology
M	Modules
MER	Merkle hash trees
n/a	not applicable
No.	Number
OCSP	Online Certificate Status Protocol
Par.	Paragraph
PKCS	Public Key Cryptographic Standard
PKI	Public Key Infrastructure
PP-0049	Identifier of the [ACMPP]
RC2	Rivest Cipher 2
resp.	respectively
RFC	Request for Comments
RMI	Remote Method Invocation
RPC	Remote Procedure Call
S	Interfaces
SASL	Simple Authentication and Security Layer

Abbreviation	Meaning
SCVP	Server-based Certification Validation Protocol
Sig	Signature
SigG	Signaturgesetz
SigV	Signaturverordnung
SOAP	Simple Object Access Protocol
SQL	Structured Query Language
SSCD	Secure Signature Creation Device
ST	Security Target
TCP	Transmission Control Protocol
TLS	Transport Layer Security
TOT	Target of Testing
TR	Technische Richtlinie
TSP	Time Stamp Protocol
USB	Universal Serial Bus
WSDL	Web Services Description Language
XAIP	XML-based Archive Information Package
XML	Extensible Markup Language
XSD	XML Schema Description

4. The Test Cases for Conformity Level 1 – Functional Conformity

4.1 Tests for all products

4.1.1 A-01 – Middleware modules should be realised as separate modules

Identifier	A-01
Requirement	M1:A3.2-1 M1:A3.1-1
Test Purpose	The test shall verify that the middleware or middleware components runs as independent applications or independent (functionally delimited) parts of an application on a trustworthy IT system. They are neither a logical nor functional component of upstream IT specialist applications and can be replaced by new, functionally compatible implementations at any time.
Configuration	CONFIG_Common
Pre-test conditions	• The middleware documentation is available

Step	Test sequence	Expected Results	Observations
1.	Check the definition of the modules in the middleware documentation. Check especially the interface definitions and whether there is a guidance for upgrading the modules to a new product version.	The middleware is based on modular components, which can be replaced by new implementations or there are explanations why this is not necessary. The interfaces and an upgrade strategies are documented.	
2.	Check whether the IT system is trustworthy on which the module is implemented. For this purpose the vendor could provide a specially hardened system or could assume a specially hardened system. The test fails, if no settings for the baseline system are assumed or already provided.[3]	There are vendor statements about the trustworthy IT system which serves as a platform for the execution of the modules.	
3.	Check the TOT and/or the user manual, whether the Modules are neither a logical or functional component of an upstream IT specialized applications.	The Modules are neither a logical nor functional component of upstream IT specialist applications.	
Verdict			

[3] For example, if the vendor just states that the product runs on the platform XYZ, the test fails.
If the vendor states that the products runs on the platform XYZ and a security white paper of the vendor of this platform may be considered, the test passes.

4.1.2 A-02 – XML-based Interfaces

Identifier	A-02		
Requirement	MD:A6.3-3		
Test Purpose	The test shall verify that the interfaces for the exchange of data between the middleware resp. components of the middleware that conforms to this guideline are generally described and realised by means of XML and corresponding schema definitions or comparable open, standardised data formats.		
Configuration	CONFIG_Common		
Pre-test conditions	• Test user has user manual and user guide.		
Step	**Test sequence**	**Expected Results**	**Observations**
1.	Check whether the TOT external interfaces for data exhange are described and defined using XML or comparable open, standardised data formats (e.g. take a look at the interface definitions within the annex TR-ESOR-E).	All interfaces are defined using XML or a comparable open, standardised format for data exchange.	
2.	Compare the implemented data exchange interfaces with their definitions described in the user manual or user guide.	The interfaces are implemented the way they have been defined.	
Verdict			

4.1.3 A-03 – No access without mutual authentication[4]

Identifier	A-03
Requirement	AS:A6.1-1 AS:A6.1-2 AS:A6.1-3 M3:A5.1-3 M3:A5.1-2
Test Purpose	The test shall verify (i) that any access from a source module to a target module can only take place via defined interfaces and is impossible without prior mutual authentication, (ii) that the mutual authentication between source and target module is cryptographically sufficient so that it is impossible to exchange individual components without being noticed and (iii) that it is impossible to bypass authentication mechanisms of two components by a replay attack.
Configuration	CONFIG_Common
Pre-test conditions	• Source and target module are not mutually authenticated (ii).

Step	Test sequence	Expected Results	Observations
1.	Check the user manual for information about interfaces.	The list of interfaces and authentications possibilities is stated.	
2.	Send requests to the target module (the TOT) without any identification or authentication at all.	One of the following results is expected: - A response is given that the request couldn't be executed.	
3.	Send requests to the target module (the TOT) after the valid authentication of the source module only.	One of the following results is expected: - A response is given that the request couldn't be executed.	
4.	Send requests to the target module (the TOT) after the valid authentication of source and target module.	A valid response is sent back by the target module.	
5.	After step 4 send another request to the target module (the TOT) without mutual authentication.	If there is no secured tunnel established: - A response is given that the request couldn't be executed. If there is a secured tunnel established: - A valid response is sent back by the target module.	
6.	Replace the source module by a fake. Do not take over the authentication credentials of the source module.	n/a	

[4] The following test course assumes that the mutual authentication of the entities can be separated. In fact, there are situations where for security reasons such a separation isn't possible. In such cases the test course must be anticipated.

7.	Try to establish a connection between source and target component (the TOT) without authentication.	- A response is given that the request couldn't be executed.	
8.	Try to establish a connection between source and target component (the TOT) with authentication. Try to also fake the authentication credentials of the faked source module.	- A response is given that the request couldn't be executed.	
9.	Verify that the authentication credentials of the TOT are not just username/password or other similar simple data.	Authentication credentials of the TOT bases on cryptography (e.g. certificates, Kerberos-tokens, …).	
10.	Start logging the data traffic between the TOT and another component.	The data logging process has been started.	
11.	Establish a valid and mutually authenticated connection between the two components and place a request from source to target module (TOT).	A valid connection is established and a valid answer from the TOT is received.	
12.	Close the connection of the two components.	The complete data exchange between the components has been intercepted and logged.	
13.	Replay the intercepted data in order to establish a valid authenticated connection between the attacker and the TOT.	No connection is established.	
Verdict			

4.1.3.1 A-03.1 – Mutual authenticated secure communication channel between client application and ArchiSafe-Module or an equivalent middleware interface

Identifier	A-03.1
Requirement	AF:A5.6-2 AF:A5.6-4 AF:A5.6-5 AF:A5-6-6
Test Purpose	The test shall verify whether a secure communication channel with certificate-based, mutual authentication is used for each transmission between the ArchiSafe module or an equivalent middleware interface and the XML module or the client application.
Configuration	CONFIG_ArchiSafe (includes TLS enforcement by ArchiSafe) if an ArchiSafe Module is present
Pre-test conditions	• The IT system documentation is available • If required, perform identification and authentication • Administration access to the IT systems is needed

Step	Test sequence	Expected Results	Observations
1.	Verify that the client application also use a secure channel for the communication with the S.4 interface of ArchiSafe.	The client application is configured in such a way that a secure channel with certificate-based mutual authentication will be used.	
2.	Try to store a XAIP_OK_SIG or BIN and then retrieve a new XAIP_OK_SIG or XAIP(BIN).	Data can be transmitted and the function be called. The XAIP/BIN can be stored.	
3.	Disable the authentication on the client application site.	Data encryption is not active any more on client application site. ArchiSafe or the equivalent middleware interface still requires a mutual authentication.	
4.	Try to store a XAIP_OK_SIG or BIN and then retrieve a new XAIP_OK_SIG or XAIP(BIN). Try to update an existing archive object. Try to delete an existing archive object.	No data is transmitted because no encryption tunnel is active. ArchiSafe or the equivalent middleware interface does not accept any unencrypted connection.	
Verdict			

4.1.3.2 A-03.2 – Mutual authenticated secure communication between XML module and ArchiSafe-Module or an equivalent middleware interface

Identifier	A-03.2
Requirement	AF:A5.6-2
Test Purpose	The test shall verify that when using a secure communication channel without certificate-based authentication, a transmission between the ArchiSafe module and the XML module is not possible.
Configuration	CONFIG_ArchiSafe (includes secure Channel enforcement by ArchiSafe) if an ArchiSafe Module is present
Pre-test conditions	• The IT system documentation is available. • If required, perform identification and authentication. • Administration access to the IT systems is needed. • **This test dispenses if no XML module is implemented.**

Step	Test sequence	Expected Results	Observations
1.	Verify that the client application also use a secure channel tunnel for the communication with the S.4 interface of ArchiSafe.	The client application is configured in such a way that a communication channel with certificate-based mutual authentication will be used.	
2.	Establish a communication channel without using a certificate on client application site.	A secure channel cannot be established.	
3.	Establish a communication channel without using a valid certificate on client application site.	A channel cannot be established.	
Verdict			

4.1.3.3 A-03.3 – secure communication channels are based on suitable cryptographic procedures

Identifier	A-03.3
Requirement	AF:A5.6-3
Test Purpose	The test shall verify that secure communication channels use cryptographic procedures that are strong enough to ensure data integrity and confidentiality.
Configuration	CONFIG_ArchiSafe (includes secure channel (e.g. TLS) enforcement by ArchiSafe) if an ArchiSafe Module is present
Pre-test conditions	• The middleware documentation is available • The IT system documentation is available • If required, perform identification and authentication • Administration access to the IT systems is needed

Step	Test sequence	Expected Results	Observations
1.	Verify that the client application also uses an encrypted communication tunnel for the communication with the S.4 interface of ArchiSafe.	The client application is configured in such a way that an encrypted communication tunnel with certificate-based mutual authentication will be used.	
2.	Try to establish an encrypted communication tunnel using a weak encryption algorithm (e.g. RC2, DES) on client application site.	A communication tunnel cannot be established.	
3.	Try to establish an encrypted communication tunnel using a strong encryption algorithm (e.g. AES-128) on client application site.	A communication tunnel can be established.	
4.	Try to establish an encrypted tunnel with illegal parameters in the handshake message	A tunnel cannot be established.	
5.	Try to establish an encrypted tunnel with a wrong or incomplete certificate	A tunnel cannot be established.	
6.	Try to establish an encrypted tunnel with a certificate expired.	A tunnel cannot be established.	
7.	Try to establish an encrypted tunnel with a wrong MAC algorithm.	A tunnel cannot be established.	
Verdict			

4.1.4 A-04 – Authentication procedure is resistant against replay attacks

Identifier	A-04
Requirement	AS:A6.1-3
Test Purpose	The test shall verify that it is impossible to bypass authentication mechanisms of two components by a replay attack.
Configuration	CONFIG_Common
Pre-test conditions	

Step	Test sequence	Expected Results	Observations
1.	Start logging the data traffic between the TOT and another component.	The data logging process has been started.	
2.	Establish a valid and mutually authenticated connection between the two components and place a request from source to target module (TOT).	A valid connection is established and a valid answer from the TOT is received.	
3.	Close the connection of the two components.	The complete data exchange between the components has been intercepted and logged.	
4.	Replay the intercepted data in order to establish a valid authenticated connection between the attacker and the TOT.	No connection is established.	
Verdict			

4.1.5 A-05 – Protection of communication channel and interface is robust against DoS-attacks

Identifier	A-05
Requirement	AS:A6.1-4
Test Purpose	The test shall verify that any unauthorised access to authentication or payload data during communication is reliably prevented and that the interface is implemented in such a way that denial of service (DoS) or consequential errors, such as buffer overflow or SQL injections are not possible.
Configuration	CONFIG_Common
Pre-test conditions	If required, perform identification and authentication.

Step	Test sequence	Expected Results	Observations
1.	Start logging the data traffic between the TOT and another component.	The data logging process has been started.	
2.	Establish a valid and mutually authenticated connection between the two components and place a request from source to target module (TOT).	A valid connection is established and a valid answer from the TOT is received.	
3.	Close the connection of the two components.	The complete data exchange between the components has been intercepted and logged.	
4.	Check if the logged traffic data reveals any authorisation or payload data.	No authorisation or payload data is revealed.	
5.	Automatically send a large amount of small requests to the TOT interface in a short period of time and check if its availability is affected (DoS). Use several client applications on several computers in parallel in order to completely fill the network bandwidth of at least 10 Mbit provided to the TOT.	The availability is not affected in a negative way. The TOT responses to all the requests or identify the DoS targets and block them.	
6.	Establish a valid connection between the components and place requests to the TOT with large amounts of data to provoke buffer overflows.	- The sent data is properly processed and checked for plausibility. - Invalid data is rejected - No buffer overflow will occur	
7.	Establish a valid connection between the components and place requests to the TOT with included database command sequences.	- The sent data is properly processed and checked for plausibility. - Invalid data is rejected - The included database commands are not executed	
Verdict			

4.1.6 A-06 – A secure tunnel can be maintained after successful authentication

Identifier	A-06
Requirement	M2:A6.2-1
Test Purpose	A secure tunnel can be maintained after successful authentication
Configuration	CONFIG_Common
Pre-test conditions	• Tester has access rights to the Cryptographic Module • No mutual authentication between the Cryptographic Module and the interface partner was made • M2 is configured to use a secure tunnel • The hash of XAIP_OK_SIG or XAIP(BIN) or BIN is present

Step	Test sequence	Expected Results	Observations
1.	Transfer the archival information package XAIP_OK_SIG or XAIP(BIN) or BIN or DXAIP_OK_SIG to the TOT using the interface (S1) function “VerifyRequest”. Observe the output of the interface function “VerifyResponse”.	The call of the function with this XAIP / XAIP(BIN) / BIN / DXAIP_OK_SIG as parameter is possible but a negative feedback will be received or the call of the function is not possible at all because Crypto Module declined connection.	
2.	Perform the mutual authentication.	Performing of the authentication is possible.	
3.	Transfer the archival information package XAIP_OK_SIG or XAIP(BIN) or BIN to the TOT using the interface (S1) function “VerifyRequest”.	The call of the function with this XAIP / XAIP(BIN) / BIN /DXAIP_OK_SIG as parameter is possible.	
4.	Observe the output of the interface function “VerifyResponse”.	A positive feedback will be received; no error message or error code.	
5.	Transfer the archival information package XAIP_OK or XAIP(BIN) or BIN DXAIP_OK_SIG to the TOT using the interface (S1) function “SignRequest” (if the function exists).	If the function exists, the call of the function with this XAIP / X(BIN)/ BIN/DXAIP_OK_SIG as parameter is possible.	
6.	Observe the output of the interface function “SignResponse”.	A positive feedback will be received; no error message or error code.	
7.	Transfer the hash of the archival information package	The call of the function with this hash as parameter is possible.	

	XAIP_OK or XAIP(BIN) or BIN or DXAIP_OK_SIG to the TOT using the interface (S3) function “TimestampRequest”.		
8.	Observe the output of the interface function “TimestampResponse”.	A positive feedback will be received; no error message or error code.	
9.	Transfer the archival information package XAIP_OK or r XAIP(BIN) or BIN or DXAIP_OK_SIG to the TOT using the interface (S3) function “HashRequest”.	The call of the function with this XAIP / X(BIN)/ BIN/DXAIP_OK_SIG as parameter is possible.	
10.	Observe the output of the interface function “HashResponse”.	A positive feedback will be received; no error message or error code.	
Verdict			

4.1.7 A-07 – Secure administration interfaces

Identifier	A-07
Requirement	MD:A6.1-4
Test Purpose	The test shall verify that the middleware supports secure administration and configuration.
Configuration	CONFIG_ArchiSafe
Pre-test conditions	• The middleware is installed and configured • The middleware documentation is available • The user has administration rights on the system

Step	Test sequence	Expected Results	Observations
1.	Check the middleware documentation for the possibilities of administration and configuration.	The documentation states that secure administration and configuration is possible.	
2.	Check the middleware's administration and configuration features.	The middleware supports secure administration and configuration.	
3.	Start a data traffic capture tool to record the data between client application and middleware.	Data traffic capturing is started.	
4.	Try to connect remotely to the middleware administration and configuration interface.	The credentials of an authorised user are needed to access the administration and configuration interface.	
5.	Try to log in to the middleware administration and configuration interface using the credentials of an unauthorised user.	Access is denied.	
6.	Try to log in to the middleware administration and configuration interface using the credentials of an authorised user.	Access is granted.	
7.	Change several options and save the current settings.	It is possible to change the configuration and save the new settings.	
8.	Stop the data traffic capture tool.	Data traffic capturing is stopped.	
9.	Check the captured traffic log file.	All the data that was transmitted during the administration process is encrypted.	
Verdict			

A-08 – No security breach induced by administration interfaces or components

Identifier	A-08
Requirement	MD:A7.3-16 M2:A6.3-3
Test Purpose	The test shall verify, that security characteristics of the middleware overall and of individual components, as well as the integrity and the authenticity of the stored data and documents can not compromised by an administration interface of the middleware or individual components without being noticed.
Configuration	CONFIG_Common
Pre-test conditions	

Step	**Test sequence**	**Expected Results**	**Observations**
1.	Check whether the access to administration interfaces is possible without any means of identification and authentication.	When accessing the administration interfaces, the user is asked for authentication.	
2.	Check whether any archive data can be accessed using the administrative interfaces that should not be accessible for the authenticated administrator.	No unauthorised access to any documents is possible.	
3.	Check whether any administration settings can be accessed that should not be accessible for an authenticated non-administrative user.	No unauthorised access to any administration setting is possible.	
4.	Check whether the administrative interface can still be used for administration after logging out.	After logging out of any administration interface none of its functions are available any more.	
5.	Check whether the actions performed by the administration interfaces are recorded in a log file.	The log file shows the performed administrative actions.	
6.	Check whether the administration interfaces allow altering digitally signed documents while bypassing the required cryptographic functions.	It is not possible to alter a digitally signed document while bypassing the required cryptographic functions.	
Verdict			

4.1.8 A-09 – Administration interfaces are available for authorised accounts only

Identifier	A-09
Requirement	MD:A7.3-15
Test Purpose	The test shall verify that any administration interfaces of the middleware or of any individual components are accessible to authorised accounts only.
Configuration	CONFIG_Common
Pre-test conditions	

Step	Test sequence	Expected Results	Observations
1.	Check if there is an official definition of an authorised account.	The authorised accounts are defined.	
2.	Try to access the administration interfaces without authentication.	It is not possible to access the administration interfaces without authentication.	
3.	Try to intercept the authentication of an authorised person to perform a replay attack.	The administration interfaces cannot be accessed.	
4.	Try to access the administration interfaces by guessing administrator credentials or unchanged system default credentials.	The administration interfaces cannot be accessed.	
Verdict			

4.1.9 A-10 – Additional interfaces do not compromise security

Identifier	A-10
Requirement	M3:A3.2-3
Test Purpose	The test shall verify that the implementation of additional interfaces shall not compromise the guarantee of basic security-relevant requirements (see Chapter 5).
Configuration	CONFIG_Common
Pre-test conditions	• **The test dispenses if no additional interfaces are implemented.** • User manual is present

Step	Test sequence	Expected Results	Observations
1.	Perform test cases A-4 and A-5, and check whether the additional interfaces of the TOT enables an attacker to spoof another secure module (e.g. ArchiSafe or the storage).	The additional interfaces do not provide such a capability or do even not provide the property to connect from or to other modules.	
2.	Perform test cases A-4 and A-5, and check whether the additional interfaces of the TOT enables an attacker to submit a data object or to request Evidence Records by circumventing security features.	The additional interfaces do not provide such a capability or do even not provide the property to connect from or to other modules.	
3.	Perform test cases A-4 and A-5, and check whether the additional interfaces of the TOT enables an attacker to circumvent the self-test function.	The additional interfaces do not provide such a capability.	
Verdict			

4.2 Module 1 – ArchiSafe

Pre-supposition:

A product which claims to comply with the M.1 ArchiSafe specification of this TR has to pass

- all test cases in this section and
- all test cases for the interface S.4 specified in Section 5.5.4 or prove that it supports functional analogous interfaces.

4.2.1 M.1-01 – ArchiSafe-module satisfies the requirements of PP-0049

Identifier	M.1-01
Requirement	MD:A7.3-2 M1:A3.3-1
Test Purpose	The test shall verify that the ArchiSafe module satisfies the requirements of PP-0049 [ACMPP] protection profile published by the Federal Office for Information Security (BSI).
Configuration	CONFIG_ArchiSafe
Pre-test conditions	

Step	Test sequence	Expected Results	Observations
1.	Check, whether there is a Common Criteria certificate of the ArchiSafe module in place (same product, same version) showing compliance with the PP-0049 (ACMPP) (preferable the latest version).	There is a such a certificate.	
2.	<u>If test step 1 fails:</u> Check whether the ArchiSafe module under testing (same product, same version) is currently in a Common Criteria evaluation and whether the Security Target claims compliance with the PP-0049 (ACMPP) (preferable the latest version).	The TOT is within the process of a Common Criteria evaluation (BSI Certification-ID 'BSI-DSZ-CC-####-####' has to be provided) and the ST claims compliance with ACMPP. In this case, the test case will be rated as “PASS WITH OBLIGATIONS” and the conformity report will state that the required ArchiSafe Protection Profile compliance could not yet be verified (with a reference to the running CC project). In addition an obligation will be attached to the TR-ESOR compliance certificate requiring the missing CC-certificate to be provided within the next 18 months otherwise the TR-ESOR compliance certificate will become invalid. If no evidence of an ongoing CC-evaluation can be provided, the test case will be rated as “FAIL”.	

Verdict

4.2.2 M.1-02 – ArchiSafe-module is separated and deployed on a trustworthy IT system

Identifier	M.1-02
Requirement	M1:A3.1-2 M1:A3.1-4
Test Purpose	The test shall verify that the ArchiSafe module is a component of the middleware and runs as an independent application or as an independent (functionally separated) part of an application on a trustworthy IT system.
Configuration	CONFIG_ArchiSafe
Pre-test conditions	• The middleware documentation is available • The IT system documentation is available

Step	**Test sequence**	**Expected Results**	**Observations**
1.	Check the IT system documentation about the implemented security mechanisms for the underlying platform.	There are recommendations or requirements to ensure the trustworthiness of the platform ArchiSafe is running on. Alternatively, ArchiSafe is delivered on a security enhanced platform.	
2.	Check the middleware documentation for a description of the design of the ArchiSafe module.	The ArchiSafe module is designed as an independent module or is at least functionally separated from other parts of the product.	
Verdict			

4.2.3 M.1-03 – Access to ECM storage should be claimed to be controlled by ArchiSafe module

Identifier	M.1-03
Requirement	MD:A7.3-1
Test Purpose	The test shall verify that any application access to the data on the ECM storage via the TOT (TR-ESOR Middleware) is claimed to be controlled and performed by the ArchiSafe module.
Configuration	CONFIG_ArchiSafe
Pre-test conditions	• The ArchiSafe module is installed and configured • The middleware documentation is available • The user has administration rights on the system

Step	Test sequence	Expected Results	Observations
1.	Check the middleware documentation for the description of the data storage process.	New data objects are not sent to the ECM directly but only by using the middleware function calls of the ArchiSafe module.	
2.	Check the middleware documentation for the description of the data change process.	Existing data objects are not changed on the ECM directly but only by using the middleware function calls of the ArchiSafe module.	
3.	Check the middleware documentation for the description of the data deletion process.	Existing data objects are not deleted from the ECM directly but only by using the middleware function calls of the ArchiSafe module.	
Verdict			

4.2.4 M.1-04 – Support of specified functions

Identifier	M.1-04
Requirement	AS:A5.4-1 M1:A4.0-1
Test Purpose	The test shall verify that the interface TR-ESOR-S.4 provides at least the following functions: • A function for the secure and reliable storage of archival information packages • • A function for retrieving archival information packages (in XAIP format) • A function for retrieving technical (cryptographic) Evidence Records • A function for deleting archived data • The test should verify that the interface TR-ESOR-S.4 provides the following functions, if implemented: • A function for updating archival information packages that have already been archived • A function for retrieving data elements of individual archival information packages • A function to verify an archive data object with evidence relevant data (signature, timestamp, certificate, revocation lists, OCSP-responses, ..) and technical evidence data (evidence record)
Configuration	CONFIG_ArchiSafe
Pre-test conditions	• User manual for S.4 interface or a functional analogous interface is accessible • Developer documents of S.4 interface the functional analogous interface are accessible

Step	**Test sequence**	**Expected Results**	**Observations**
1.	Check if the middleware documentation contains the description of the necessary functions.	The necessary functions are defined in the documentation.	
2.	Store anXAIP_OK_SIG or BIN using the “ArchiveSubmissionRequest” function.	The function call is possible.	
3.	Check the output of the “ArchiveSubmissionResponse” function.	The XAIP/BIN object is assigned to an AOID and returned-successfully.	
4.	If the function “ArchiveUpdateRequest” is implemented, use the “ArchiveUpdateRequest” function with the AOID from step 3 and a DXAIP_OK to change ~~the~~ data stored within the XAIP/ XAIP(BIN).	The function call is possible.	

5.	If the function “ArchiveUpdateRequest” is implemented, check the output of the “ArchiveUpdateResponse” function.	A new version ID is received.	
6.	Use the “ArchiveRetrievalRequest” function with the AOID from step 3 to retrieve a XAIP with all versions (e.g. Version Id = “all”) from the storage.	The function call is possible.	
7.	Check the output of the “ArchiveRetrievalResponse” function.	The archive data object is received in XAIP format.	
8.	Use the “ArchiveEvidenceRequest” function with the AOID from step 3 to check the XAIP / BIN authenticity and integrity concerning all versions.	The function call is possible.	
9.	Check the output of the “ArchiveEvidenceResponse” function.	If there exist only one version of the archive data object, one Evidence Record is received. Otherwise, for each existing version an Evidence Record is received.	
10.	If the “ArchiveDataRequest” is implemented, use the “ArchiveDataRequest” function with the AOID from step 3 and a valid dataLocation parameter to identify an individual data element within the XAIP or BIN.	The function call is possible.	
11.	If the “ArchiveDataRequest” is implemented, check the output of the “ArchiveDataResponse” function.	The requested data value and the original locationValue are received.	
12.	Use the “ArchiveDeletionRequest” function with the AOID from step 3 to delete the XAIP or the BIN.	The function call is possible.	
13.	Check the output of the “ArchiveDeletionResponse” function.	The XAIP or BIN has been deleted from the storage.	
14.	If the “VerifyRequest” is implemented, use the “Verify Request” function with the XAIP/BIN from from step 3 and the evidence records from step 9 to check the XAIP, the evidence relevant data and the evidence record(s).	The function call is possible.	
15.	If the “VerifyRequest” is implemented check the output of the “VerifyResponse” function.	The “VerifyRequest” is possible and returns a return code or a verification report, if ordered.	
16.	Check the results of the test cases S.4.1-01 – S.4.1-07 S.4.2-01 – S.4.2-03 S.4.3-02	The tests are performed successfully	

	S.4.4-02, S.4.4-03 S.4.5-01 – S.4.5-04 S.4.6-01 or functional analogous test cases		
Verdict			

4.2.5 M.1-05 – Using interfaces S.1 and S.6 is possible

Pre-supposition:

A product which claims to functionally comply with the interfaces specification S.1 and S.6 of this TR or part of it, has to pass the following test case or part of it or prove that it supports functional analogous interfaces.

Identifier	**M.1-05**
Requirement	M1:A3.2-2
Test Purpose	The test shall verify that the ArchiSafe module is able to access the other modules of the middleware via dedicated interfaces as described in the annexes TR-ESOR-M.2, TR-ESOR-M.3 and TR-ESOR-S of this technical guideline.
Configuration	CONFIG_ArchiSafe
Pre-test conditions	• The tests of test case M.1-01 have been successfully completed • The middleware documentation is available • **The test dispenses if the pre-supposition is not valid.**

Step	**Test sequence**	**Expected Results**	**Observations**
1.	Check if the ArchiSafe documentation contains the description of how to connect to the interface S.1.	The interface is described in the documentation.	
2.	Check if the ArchiSafe documentation contains the description of how to connect to the interface S.6.	The interface is described in the documentation.	
3.	Check if it is possible for the ArchiSafe module to communicate with the Crypto Module via the S.1 interface.	Communication is possible.	
4.	Check if it is possible for the ArchiSafe module to communicate with the ArchiSig module via the S.6 interface.	Communication is possible.	
Verdict			

4.2.6 M.1-06 – Comprehensive and configurable options for logging

Identifier	M.1-06
Requirement	M1:A4.0-3
Test Purpose	The test shall verify that the ArchiSafe module offers comprehensive and configurable options for logging any access to the archive.
Configuration	CONFIG_ArchiSafe
Pre-test conditions	• The ArchiSafe module is installed and configured • The user has administration rights on the system

Step	Test sequence	Expected Results	Observations
1.	Check the user manual of the software for logging options.	Comprehensive and configurable logging options are described in the user manual.	
2.	Configure the log function to the most comprehensive level.	Any kind of access to the archive will be logged to the log file.	
3.	Store an XAIP_OK_SIG or BIN using the “ArchiveSubmissionRequest” function.	The function call is possible. The XAIP / BIN object is assigned an AOID and stored successfully.	
4.	If implemented, use the “ArchiveUpdateRequest” function with the AOID from step 3 and a DXAIP_OK to change the data contained within the XAIP or XAIP(BIN).	The function call is possible. A new version ID is received.	
5.	Use the “ArchiveRetrievalRequest” function with the AOID from step 3 to retrieve the XAIP/XAIP(BIN) from the storage.	The function call is possible. The archive data object is received in XAIP format.	
6.	Use the “ArchiveEvidenceRequest” function with the AOID from step 3 to check the XAIP / BIN authenticity and integrity for all versions.	The function call is possible. If there exist only one version of the archive data object, one Evidence Record is received. Otherwise, for each existing version an Evidence Record is received.	
7.	If implemented, use the “ArchiveDataRequest” function with the AOID from step 3 and the dataLocation parameter to identify an individual data element within the XAIP / BIN.	The function call is possible. The requested data value and the original locationValue are received.	
8.	Use the “ArchiveDeletionRequest” function with the AOID from step 3 to delete the XAIP / BIN.	The function call is possible.	
9.	Check the deletion by calling the	The “ArchiveRetrievalResponse” indicates that no stored object	

	"ArchiveRetrievalRequest " with AOID from step 3.	with corresponding AOID can be found in the storage	
10.	If the "VerifyRequest" is implemented, use the "Verify Request" function with the XAIP / BIN from step 3 and the evidence records from step 6 to check the XAIP/BIN, the evidence relevant data and the evidence record(s).	The function call is possible.	
11.	If the "VerifyRequest" is implementedm, check the output of the "VerifyResponse" function.	The "VerifyRequest" is possible and returns a return code or a verification report, if ordered.	
12.	Check the log file for logs of all the access procedures from the previous steps.	The log file contains all the access procedures from the previous steps and also the return codes (error, success) and actual return values.	
Verdict			

4.2.7 M.1-07 – Access to log files is possible by authorized persons only

Identifier	M.1-07
Requirement	M1:A4.0-4
Test Purpose	The test shall verify that only authorised persons are able to access the log files that have been created by the ArchiSafe module.
Configuration	CONFIG_ArchiSafe
Pre-test conditions	• The ArchiSafe module is installed and configured • The user has administration rights on the system

Step	Test sequence	Expected Results	Observations
1.	Check the vendor documentation whether there is a description how to restrict the access to the log records.	There is such a description or the documentation refers to the access control mechanism of the underlying platform.	
2.	Check the vendor documentation whether there are recommendations regarding the access control restrictions for the log files.	There are such recommendations. It is recommended that only the authorized persons shall be able to access (read) the log files. Nobody shall be able to modify the log files. Only administrators are allowed to delete the log files after archiving or after the end of use.	
3.	Configure access restrictions as recommended in the guidance.	Successfully possible.	
4.	Verify that an unauthorized person is not able to access the log records. Please take all recommended security mechanisms into account, also the organizational and physical ones.	Access is not possible.	
Verdict			

4.2.8 M.1-08 – Changing metadata or data objects results in a new version of stored XAIP or BIN

Pre-supposition:

A product which claims to to comply with the update functionality according to M.1-04 and S.4.2-01 “ArchiveUpdateRequest” of this TR has to pass the following test case or prove that it supports functional analogous functions.

Identifier	**M.1-08**
Requirement	M1:A4.2-6
Test Purpose	The test shall verify that any changes of metadata or data objects within an XAIP or BIN is based on the principles defined in the TR documentation.
Configuration	CONFIG_ArchiSafe
Pre-test conditions	• The middleware documentation is available • The user has administration rights on the system • Test case S.4-24 has been tested successfully • **The test dispenses if the pre-supposition is not valid.**

Step	Test sequence	Expected Results	Observations
1.	Check the middleware documentation for the procedure of the update process. If the update functionality is supported it is important that per update (version) a new version manifest will be created, new/updated data will be added, for “removed” data just the links in the new version manifest will be removed – the data keeps stored in the XAIP / BIN.	The data update function is documented as defined in the TR.	
2.	Store an XAIP_OK or BIN using the interface function “ArchiveSubmissionRequest”.	The call is successful, a valid AOID is returned	
3.	Use the “ArchiveRetrievalRequest” with returned AOID from step 2 to request an XAIP_OK or an XAIP(BIN).	The call is successful.	
4.	Use the “ArchiveUpdateRequest” with the returned AOID and a DXAIP_OK to create a new version with updated metadata in the archived XAIP_OK / XAIP(BIN)	The call is possible, no error is returned	
5.	Use the “ArchiveRetrievalRequest” with returned AOID to request an XAIP_OK and check if the version	The call is successful, the version manifest has been changed	

	manifest has been changed.		
6.	Use the “ArchiveUpdateRequest” with the returned AOID and the returned VersionID of step 5 to create a new version with updated data objects in the archived XAIP_OK / XAIP(BIN)..	The call is possible, no error is returned	
7.	Use the “ArchiveRetrievalRequest” with returned AOID to request an XAIP_OK and check if the version manifest has been changed.	The call is successful, the version manifest has been changed	
Verdict			

4.2.9 M.1-09 – ArchiSafe-module should be capable of serving and separating multiple clients

Identifier	M.1-09
Requirement	MD:A6.1-3
Test Purpose	The test should check whether the middleware is able to manage multiple clients and separate the different clients' data.
Configuration	CONFIG_ArchiSafe
Pre-test conditions	• The middleware documentation is available • If required, perform identification and authentication

Step	Test sequence	Expected Results	Observations
1.	Check the middleware documentation for the management of multiple clients.	It is possible to manage multiple clients simultaneously while storing their data separately.	
2.	Authenticate with valid user credentials of client A.	The authentication is successful.	
3.	Store an XAIP_OK or BIN using the interface function “ArchiveSubmissionRequest”.	The call of the function with this XAIP / BIN as a parameter is possible.	
4.	Observe the output of the interface function “ArchiveSubmissionResponse”.	A positive feedback is received. No error message or error code is returned. An AOID is assigned.	
5.	Authenticate with valid user credentials of client B.	The authentication is successful.	
6.	Attempt to get an “ArchiveRetrievalRequest” with the AOID from client A.	The access will be denied.	
7.	Repeat the test sequence storing first an XAIP_OK or BIN assigned to client B and then attempt to access stored data (“ArchiveRetrievalRequest”) with an authentication of client A.	The access will be denied.	
Verdict			

4.2.10 M.1-10 – ArchiSafe-Module is thread safe

Identifier	M.1-10
Requirement	AF:A5.6-9 M1:A4.0-6
Test Purpose	The test shall verify that the ArchiSafe module can process several transactions simultaneously.
Configuration	CONFIG_ArchiSafe
Pre-test conditions	• The middleware documentation is available • The IT system documentation is available • The application documentation is available • If required, establish a session with the TOT in order to perform the following tests • If required, perform identification and authentication • A sufficient amount of XAIPs or BINs have already been stored on the ECM storage to perform the technical tests

Step	Test sequence	Expected Results	Observations
1.	Use a number of "ArchiveRetrievalRequests" with valid AOIDs to request a number (at least 20) of XAIPs from one client application.	The function calls with the given AOIDs are possible.	
2.	Observe the output of the severeal interface functions "ArchiveRetrievalResponse".	Positive feedbacks are received. No error messages or error codes occur. The requested XAIPs are retrieved successfully.	
3.	If possible use a number of "ArchiveRetrievalRequests" with valid AOIDs to request a number (at least 20) of XAIPs from at least 2 client applications simultaneously. (Request the same XAIPs from both clients)	The function calls with the given AOIDs are possible.	
4.	Observe the output of the several interface functions "ArchiveRetrievalResponse".	Positive feedbacks are received. No error messages or error codes occur. The requested XAIPs are retrieved successfully by both client applications.	
Verdict			

4.2.11 M.1-11 – Access rights are enforced for individual archive objects

Identifier	M.1-11

Requirement	M1:A4.0-6 M1:A5.0-1 M1:A5.0-3
Test Purpose	The test shall verify that client software can only access archive objects for which it has access rights. This is also stringently enforced when several archival information packages are requested simultaneously and, as applicable, there are only access rights to a few of them.
Configuration	CONFIG_ArchiSafe (including at least two different and separated clients configured)
Pre-test conditions	• If required, perform identification and authentication • If required, the tester has to manually simulate access requests as if they were issued by client applications • The call of the function "ArchiveSubmissionRequest" by a client application A with a XAIP_OK or BIN as a parameter is possible. A positive feedback is received. No error message or error code occurs. An AOID A1 is assigned. • The call of the function "ArchiveSubmissionRequest" by a client application A with another XAIP_OK or BIN as a parameter is possible. A positive feedback is received. No error message or error code occurs. An AOID A2 is assigned. • The call of the function "ArchiveSubmissionRequest" by a client application B with a XAIP_OK or BIN as a parameter is possible. A positive feedback is received. No error message or error code occurs. An AOID B1 is assigned.

Step	Test sequence	Expected Results	Observations
1.	By using client application **A**: Using the interface function "ArchiveRetrievalRequest" and the AOID **A1** to request the XAIP.	The call of the function with this AOID as a parameter is possible.	
2.	Observe the output of the interface function "ArchiveRetrievalResponse".	A positive feedback is received. No error message or error code occurs. An XAIP is received.	
3.	By using client application **B**: Using the interface function "ArchiveUpdateRequest", if implemented, and the AOID **A1** with any DXAIP_OK as update data to update the XAIP or XAIP(BIN).	The call of the function with this AOID as a parameter is possible.	
4.	Observe the output of the interface function "ArchiveUpdateResponse".	A negative feedback is received. An error message or error code occurs because access is denied. The XAIP / XAIP(BIN) is not updated.	
5.	By using client application **B**: Using the interface function "ArchiveRetrievalRequest" and the AOID **A1.** to request the XAIP/XAIP(BIN).	The call of the function with this AOID as a parameter is possible.	
6.	Observe the output of the interface function "ArchiveRetrievalResponse".	A negative feedback is received. An error message or error code occurs because access is denied. No XAIP is received.	
7.	By using client application **B**: Using the interface	The call of the function with this AOID as a parameter is possible.	

	function “ArchiveDeletionRequest“ and the AOID **A1** to delete the XAIP or BIN.		
8.	Observe the output of the interface function “ArchiveDeletionResponse”.	A negative feedback is received. An error message or error code occurs because access is denied. The XAIP / BIN is not deleted.	
9.	By using client application **A**: Using the interface function “ArchiveRetrievalRequest“ and the AOID **A1**, **A2** and **B1** to request the XAIPs.	The call of the function with this AOID as a parameter is possible.	
10.	Observe the output of the interface function “ArchiveRetrievalResponse”.	A mixed feedback is received. The XAIP's **A1** and **A2** could be retrieved, for **B1** an error was received.	
11.	By using client application **B**: Using the interface function “ArchiveRetrievalRequest“ and the AOID **A1**, **A2** and **B1** to request the XAIPs.	The call of the function with this AOID as a parameter is possible.	
12.	Observe the output of the interface function “ArchiveRetrievalResponse”.	A mixed feedback is received. The XAIP **B1** could be retrieved, for **A1** and **A2** an error was received.	
13.	Try to use a client application **C**, which is not an authorized archive application, to submit a XAIP or BIN, to update a XAIP or XAIP(BIN), to retrieve a XAIP or to delete a XAIP or BIN of another client.	A negative feedback is received. An error message or error code occurs. Access to the middleware and the storage is denied in any case.	
Verdict			

4.3 Module 2 – Crypto Module

A product which claims to comply with the M.2 Crypto Module specification of this TR has to pass

- all test cases in this section and
- all test cases for the interface S.1 and S.3 specified in section 5.5.1 and 5.5.3 respectively or prove that it supports functional analogous interfaces.

4.3.1 M.2-01 – Crypto Module is a signature application component according to § 17 Par. 2 SigG

Identifier	M.2-01		
Requirement	MD:A7.3-4		
Test Purpose	The Cryptographic Module fulfils the requirements of a signature application component pursuant to § 17 Sec. 2 SigG at a minimum.		
Configuration	CONFIG_Common		
Pre-test conditions	• User manual is present		
Step	**Test sequence**	**Expected Results**	**Observations**
1.	Check the user manual and related documentation, if there is described that the Cryptographic Module fulfils the requirements of a signature application component pursuant to § 17 Par. 2 SigG at a minimum.	In the user manual there is a confirmation, that the Cryptographic Module fulfil the requirements of a signature application component pursuant to § 17 Par. 2 SigG at a minimum. This means that there is a certification and confirmation according to SigG or there is a declaration of the vendor according to § 17 Par. 4 SigG.	
Verdict			

4.3.2 M.2-02 – Crypto Module may be SSCD according to § 17 Par. 1 SigG

Pre-supposition:

A product which claims to comply the M.2 Crypto Module specification of this TR and which intends to generate qualified signatures by itself has to pass the following test case.

Identifier	**M.2-02**
Requirement	MD:A7.3-5
Test Purpose	If the module is intended to create qualified electronic signatures itself, the TOT fulfils the requirements for a secure signature creation device pursuant to § 17 Par. 1 SigG.
Configuration	CONFIG_Common
Pre-test conditions	• User manual for TR-ESOR M.2 is present • T**he test dispenses if the pre-supposition is not valid.**

Step	**Test sequence**	**Expected Results**	**Observations**
1.	Check, whether the module is able to create qualified signatures itself.	The module may be able to create qualified electronic signatures.	
2.	Check, whether for the software or hardware units which are supposed to create qualified electronic signatures or the complete TOT there exists a confirmation that it is an approved secure signature creation device pursuant to § 17 Par. 1 SigG.	Such a confirmation exists for the components, which are supposed to create qualified electronic signatures.	
Verdict			

4.3.3 M.2-03 – Cryptographic algorithms must be exchangeable

Identifier	M.2-03
Requirement	MD:A7.3-6 M2:A3.2-1
Test Purpose	The algorithms and parameters of the Cryptographic Module that are suitable for security can be exchanged in a quick and uncomplicated manner.
Configuration	CONFIG_Common
Pre-test conditions	• User has administrator rights on the system • User manual is present

Step	**Test sequence**	**Expected Results**	**Observations**
1.	Check, whether a hash-algorithm and parameters can be changed in a quick and uncomplicated manner.	The hash-algorithm and parameters can be changed in a quick and uncomplicated manner.	
2.	Check, whether a signature-algorithm and parameters can be changed in a quick and uncomplicated manner.	The signature-algorithm and parameters can be changed in a quick and uncomplicated manner.	
Verdict			

4.3.4 M.2-04 – Crypto Module should fulfil the requirements of TR-03112

Identifier	M.2-04		
Requirement	MD:A7.3-7		
Test Purpose	The interfaces of the Cryptographic Module should fulfil the requirements of the BSI Technical Guideline TR-03112 (eCard-API-Framework).		
Configuration	CONFIG_Common		
Pre-test conditions	• User manual is present		
Step	**Test sequence**	**Expected Results**	**Observations**
1.	Check, if at least the external interfaces of the Cryptographic Module are implemented in software.	The external interfaces may be implemented in software (e.g. libraries, API).	
2.	If step 1 passed: Check whether there is a conformity statement to TR-03112.	A conformity statement to TR-03112 exist.	
Verdict			

4.3.5 M.2-05 – Crypto Module should be certified according to SigG

Identifier	M.2-05		
Requirement	M2:A3.3-1		
Test Purpose	The Cryptographic Module should be certified pursuant to SigG and SigV. The test checks whether the Cryptographic Module is certified accordingly.		
Configuration	CONFIG_Common		
Pre-test conditions	• User manual and developer documents are present		
Step	**Test sequence**	**Expected Results**	**Observations**
1.	Check user manual if the product has certifications pursuant to the Signature Act.	The product that provides the functions of the Cryptographic Module has certification.	
Verdict			

4.3.6 M.2-06 – Random number generators fulfil the BSI requirements

Identifier	M.2-06		
Requirement	M2:A4.1-2		
Test Purpose	The random number generators used by the Cryptographic Module fulfil the requirements set forth in the BSI Technical Guidelines [TR 03116] and [TR 02102] pursuant to [AIS 20] for pseudo random number generators or according to [AIS 31] for physical random number generators.		
Configuration	CONFIG_Common		
Pre-test conditions	• User manual and developer documents are present		
Step	**Test sequence**	**Expected Results**	**Observations**
1.	Check user manual and developer documents, whether the random number generators fulfil the requirements defined by set for the BSI Technical Guidelines [TR 03116] and [TR 02102] pursuant to [AIS20] for pseudo random number generators or according to [AIS 31] for physical random number generators.	The random number generators fulfil the defined requirements set.	
Verdict			

4.3.7 M.2-07 – Support of Hash functions

Identifier	M.2-07
Requirement	M2:A4.2-1 M2:A4.2-2 M2:A4.2-3 M2:A5.2-1
Test Purpose	The Cryptographic Module shall have functions to calculate hash values for information packages. In doing so, the requirements for hash procedures shall be fulfilled.[5]
Configuration	CONFIG_Common
Pre-test conditions	• The list of hash algorithms and parameters recommended by the Federal Office for Information Security and the Federal Network Agency is accessible • User manual is present

Step	Test sequence	Expected Results	Observations
1.	Check the user manual for the hash algorithms which are used by the Cryptographic Module. The Cryptographic Module shall support at least two hash algorithms which have been assessed by the Federal Office of Information Security and the Federal Network Agency as suitable for security and published.	The used hash algorithms are in the list of the recommended algorithms.	
2.	Check the user manual for the supported hash algorithms.	The Cryptographic Module supports all previously used hash algorithms.	
Verdict			

[5] Exclusively those hash algorithms and parameters recommended by the Federal Office for Information Security and the Federal Network Agency shall be used to form hash values. However, the Cryptographic Module shall continue to support all hash algorithms previously used by the Cryptographic Module in order to enable verification of hash values generated in the past according to [ALGCAT] and [TR-ESOR-ERS], chapter 5.2.1).

4.3.8 M.2-08 – Crypto Module uses recommended algorithms for generating signatures

Pre-supposition:

A product which claims to comply the M.2 Crypto Module specification of this TR and which intends to generate signatures by itself has to pass the following test case.

Identifier	**M.2-08**
Requirement	M2:A4.3-1
Test Purpose	The test evaluates if the algorithms implemented by the Cryptographic module for generating signatures comply with the current version of the algorithm catalogue "Geeignete Algorithmen zur Erfüllung der Anforderungen nach §17 Abs. 1 bis 3 SigG vom 22. Mai 2001 in Verbindung mit Anlage 1 Abschnitt I Nr. 2 SigV vom 22. November 2001" [Suitable algorithms to fulfil requirements accordant to §17 Par. 1 through 3 SigG from 22 May 2001 together with Annex 1 Section I No. 2 SigV from 22 November 2001] [ALGCAT].
Configuration	CONFIG_Common
Pre-test conditions	• User manual and developer documents are present • **The test dispenses if the pre-supposition is not valid.**

Step	**Test sequence**	**Expected Results**	**Observations**
1.	Check the user manual, whether the Crypto-Module complies with the current version of the algorithm catalogue.	The Cryptographic Module complies with the current version of the algorithm catalogue.	
2.	Check the developer documents, whether the requirements from Chapter 4.3 of annex TR-ESOR M2 are implemented for generating signatures.	The requirements from Chapter 4.3 are implemented.	
Verdict			

4.3.9 M.2-09 – Crypto Module supports canonicalisation for the verification of XML signatures

Identifier	M.2-09
Requirement	M2:A4.4-2 M2:A4.4-4
Test Purpose	Support of canonicalisation procedures for the verification of of XML signatures. The support of canonicalisation procedure C14N - Canonical XML Version 1.0 [C14N] - is supported at a minimum. Note: if the TOT doesn't support XML signatures the test case can be passed as fulfilled.
Configuration	CONFIG_Common
Pre-test conditions	• User manual is present • Security architecture design is present

Step	Test sequence	Expected Results	Observations
1.	Check the user manual and security architecture design, whether the support of canonicalisation procedures at least for the verification of signatures of XML contents by the Cryptographic Module is given.	The support of canonicalisation procedures for the verification of signatures of XML contents by the Cryptographic Module is present.	
2.	Check the developer documents for information about how the canonicalisation procedure was implemented.	The implementation of the canonicalisation procedure support C14N – Canonical XML Version 1.0 [C14N] at a minimum. C14N11 – Canonical XML Version 1.1 [C14N11], C14N20 – Canonical XML Version 2.0 [C14N20] and EC14N - Exclusive XML Canonicalization – should also be supported.	
3.	Generate a signed XML, e.g. a signed XAIP or BIN. It is not necessary to produce the signature with this Crypto Module.	---	
4.	Verify the signature of the XAIP / BIN.	The verification result should show a positive result. Signature is valid.	
5.	Modify the signed XAIP/BIN in such a way so that it is not canonicalised (e.g. by entering empty lines and spaces between the XML tags). Do not modify or remove the signature.	---	
6.	Verify the signature of the XAIP/BIN.	The verification result should show a positive result. Signature is valid.	

Verdict

4.3.10 M.2-10 – Canonicalisation procedures do not change the content data

Identifier	M.2-10
Requirement	M2:A4.4-3
Test Purpose	The implemented canonicalisation procedures shall not change the content data. Note: if the TOT doesn't support "ArchiveSubmissionRequests" with XML data as parameters the test case can be passed as fulfilled.
Configuration	CONFIG_Common
Pre-test conditions	• XML-data with empty tags, additional white spaces, wrong order of XML-tags and signature is present, e.g. from test case M.2-09.

Step	Test sequence	Expected Results	Observations
1.	Using the interface function "VerifyRequest" send XML-data to the Cryptographic Module.	The sending of the XML-data is possible.	
2.	Observe the output of the interface function "VerifyResponse".	A positive feedback will be received; no error message or error code.	
3.	Check the field „responseData".	The field „responseData" contains • no XML-data but only the results of the verification OR • XML-data and the results of the verification	
4.	If XML-data are returned, compare the received XML-data with original XML data.	The contents of both XML-files are equal (unmodified) or the XML-data is modified (assumed: canonicalised).	
5.	Check the result of canonicalisation whether the unmodified and the modified XML-data is equal related to the content and mappable for XML syntax and XSD used.	The canonicalisation is correct.	
Verdict			

4.3.11 M.2-11 – XML-Signatures follow the recommendations of RFC3275

Identifier	M.2-11
Requirement	M2:A5.1-3 M2:A5.3-8
Test Purpose	Electronic signatures of XML data will be generated in the following format and follow the basic recommendations in [Common PKI] (Part 8): XML Signature Standard [RFC3275]. Note: if the TOT doesn't support XML signatures the test case can be passed as fulfilled.
Configuration	CONFIG_Common
Pre-test conditions	• User manual is present • Developer documents are present

Step	**Test sequence**	**Expected Results**	**Observations**
1.	If the product claims to be able to generate electronic signatures, check the user manual and developer documents, if electronic signatures of XML data are generated according to XML Signature Standard [RFC 3275].	Electronic signatures of XML data are generating according to XML Signature Standard [RFC3275]. Alternatively: the TOT is certified according to BSI TR-03112.	
2.	Check the user manual and developer documents, if the canonicalisation procedure is used when using RFC 3275 format.	The canonicalisation procedure is used when using RFC 3275 format.	
Verdict			

4.3.12 M.2-12 – Reliable verification of electronic signatures

Identifier	M.2-12
Requirement	M2:A5.1-7
Test Purpose	The Cryptographic Module that conforms to this Guideline shall provide functions for the reliable verification of electronic signatures. The signature verification function of the Cryptographic Module supports the signature data formats XML Signature and CMS Signature at a minimum.
Configuration	CONFIG_Common
Pre-test conditions	• User manual is present

Step	Test sequence	Expected Results	Observations
1.	Check the user manual if the Cryptographic Module provides a function for the reliable verification of electronic signatures.	The Cryptographic Module provides such a function.	
2.	Check the user manual for information about which signature data formats are supported by the Cryptographic Module.	The Cryptographic Module supports the XML Signature Standard [RFC3275] and the Cryptographic Message Syntax (CMS) [RFC3852]	
3.	Use “VerifyRequest” - function to verify an XML signature	Verification of XML signatures are supported by that function.	
4.	Compare signature verification results of the Cryptographic module with results of a common certified tool or product. Alternatively: the TOT is certified according to BSI TR-03112. Then, this test step is not required.	The signature verifications offer identical results OR the product is certified according to BSI TR-03112.	
5.	Use “VerifyRequest” - function to verify a CMS signature	Verification of CMS signatures are supported by that function.	
6.	Compare signature verification results of the Cryptographic module with results of a common certified tool or product. Alternatively: the TOT is certified according to BSI TR-03112. Then, this test step is not required.	The signature verifications offer identical results OR the product is certified according to BSI TR-03112.	
Verdict			

4.3.13 M.2-13 – Crypto-Module shall have function to validate certificate chains

Identifier	M.2-13
Requirement	M2:A5.1-17
Test Purpose	The Cryptographic Module shall have a function to validate certificate chains in order to verify the integrity of archived certificate chains and archived packages (see [RFC5280] Section 6 and [TR-ESOR-M.3]). The list of trusted certificates can be configurable.
Configuration	CONFIG_Common
Pre-test conditions	• Certificate of a Certification Service Provider is present

Step	Test sequence	Expected Results	Observations
1.	Sign the XAIP_OK or DXAIP_OK or BIN archival information package using a valid and not expired certificate issued by a Trust Center.	The signed XAIP_OK / DXAIP_OK / BIN was created successfully.	
2.	Transfer the signed XAIP_OK or DXAIP_OK or BIN to the TOT using the interface function „VerifyRequest“.	The call of the function with this XAIP / DXAIP_OK / BIN as parameter is possible.	
3.	Observe the output of the interface function “VerifyResponse”.	A positive feedback will be received; the signature has been verified.	
4.	Check the verification results whether the certificate used for the signature of the XAIP / DXAIP_OK / BIN was verified.	The certificate used for the signature was verified. The verification results are included.	
5.	Check the verification results whether the CA certificate used for the signature of the certificate was verified.	The CA certificate was verified. The verification results are included. There must be an indication that this certificate is a strusted root CA certificate.	
6.	Check the user manual if the list of trusted certificates may be configured.	The list of trusted certificates may be configured.	
7.	Perform a test with a configured list of trusted certificates	The Cryptographic Module can check a configured list of trusted certificates.	
Verdict			

4.3.14 M.2-14 – Verification of signatures yields standardised and comprehensive verification report

Identifier	M.2-14
Requirement	M2:A5.1-10 M2:A5.1-11 M2:A5.1-12
Test Purpose	The Cryptographic Module is able to generate signature verification results in standardised formats. The Cryptographic Module shall be able to return the signature verification results, including related certificate information. The Cryptographic Module shall offer a function that is able to validate user certificates for electronic signatures. Verification shall be complete up to a trustworthy root.
Configuration	CONFIG_Common
Pre-test conditions	• User manual is present

Step	Test sequence	Expected Results	Observations
1.	Check the user manual whether the cryptographic module has a function that is able to demonstrably verify the presence and validity status of user certificates for electronic signatures at the time of signature creation.	The cryptographic module provides such a function.	
2.	Transfer the archival information package XAIP_OK_SIG and / orBIN to the TOT using the interface function “VerifyRequest”.	The call of the function with this signed object as parameter is possible.	
3.	Observe the output of the verify- function “VerifyResponse”.	A positive feedback will be received; no error message or error code.	
4.	Check, whether verification information is missing. The complete verification information of the signature, the certificate and all certificates back to a trustworthy root CA must be present.	All the signature verification results, including related certificate information, are returned without changes to the module making the request.	
5.	Check the format for the verification results. Check the user guidance to determine the format used.	The results are documented in a standardized format. Preferably the “VerificationReport” of the eCard-API-Framework is used.	
6.	If implemented, transfer the archival information package XAIP_OK_SIG and / or BIN to the TOT using the interface function “VerifyRequest” and asking for a `ReturnVerificationReport`.	The call of the function with this signed object as parameter is possible.	
7.	Observe the output of the interface function	A positive feedback will be received; no error message or error	

	"VerifyResponse".	code.	
8.	Check, whether verification information is missing. The complete verification information of the signature, the certificate and all certificates back to a trustworthy root CA must be present.	All the signature verification results, including related certificate information, are returned without changes to the module making the request. A `ReturnVerificationReport`-element according **[OASIS VR], [eCard-2]** and **[TR-ESOR-VR]** is returned.	
9.	Check the format for the verification results. Check the user guidance to determine the format used.	The results are documented in a standardized format of a "VerificationReport" of the eCard-API-Framework (**[OASIS VR], [eCard-2]** and **[TR-ESOR-VR])**.	
10.	If implemented, transfer the archival information package XAIP_OK_SIG and / or BIN together with at least one evidence record to the TOT using the interface function "VerifyRequest" and asking for a `ReturnVerificationReport`.	The call of the function with this signed object as parameter is possible.	
11.	Observe the output of the interface function "VerifyResponse".	A positive feedback will be received; no error message or error code.	
12.	Check, whether verification information is missing. The complete verification information of the signature, the certificate and all certificates back to a trustworthy root CA must be present.	All the signature verification results, including related certificate information, and evidence record verification are returned without changes to the module making the request. At least one `ReturnVerificationReport`-element according **[OASIS VR], [eCard-2]** and **[TR-ESOR-VR]** is returned.	
13.	Check the format for the verification results. Check the user guidance to determine the format used.	The results are documented in a standardized format of a "VerificationReport" of the eCard-API-Framework (**[OASIS VR], [eCard-2]** and **[TR-ESOR-VR])**.	
Verdict			

4.3.15 M.2-15 – Protecting private keys

Identifier	M.2-15
Requirement	M2:A6.1-2 M2:A6.2-5
Test Purpose	Private keys stored in the Cryptographic Module shall not be accessible for unauthorised users.
Configuration	CONFIG_Common
Pre-test conditions	

Step	Test sequence	Expected Results	Observations
1.	Check vendor documentation whether the Crypto Module is able to store private keys longer than just for signature operations.	The Crypto Module may have such a function. If not, this test case is finished and considered to be passed.	
2.	Check vendor documentation whether the Crypto Module provides functions to directly and explicitly access (read) the private keys or to perform cryptographic operations with these keys.	The Crypto Module may have such a function. If not, this test case is finished and considered to be passed.	
3.	Verify each of these functions whether an identification and authentication is required prior to actual execution of the function.	Every function requires at least authentication prior execution.	
4.	Check vendor documentation for information about where keys are stored in the system.	The keys are stored in a protection system implemented as a hardware solution, e.g. USB-tokens or a smart card. If yes, this test case is finished and considered to be passed.	
5.	Check vendor documentation for information about how keys are stored software-based (typically as file).	The Public Key Cryptography Standard #12 [PKCS#12] format is used to store keys and X.509v3 certificates.	
Verdict			

4.3.16 M.2-16 – Suitability of cryptographic algorithms should be defined by policy file

Identifier	M.2-16		
Requirement	M3:A5.3-2		
Test Purpose	Check whether the validity periods of hash and signature algorithms are stored and managed in the form of a policy file.		
Configuration	CONFIG_Common		
Pre-test conditions	• User manual is present		
Step	**Test sequence**	**Expected Results**	**Observations**
1.	Check the user manual how the validity periods of hash and signature algorithms are stored and managed.	The validity periods of hash and signature algorithms should be stored and managed in the form of a policy file.	
Verdict			

4.3.17 M.2-17 – Protect its own security

Identifier	M.2-17		
Requirement	M2:A6.1-3 M2:A6.1-4 M2:A6.2-2		
Test Purpose	Check whether the Cryptographic Module includes a function to verify its own integrity as internal defence against manipulation.		
Configuration	CONFIG_Common		
Pre-test conditions	• Developer documents • Design documents		
Step	**Test sequence**	**Expected Results**	**Observations**
1.	Check the vendor documentation for information whether the Cryptographic Module includes a function to verify its own integrity.	The Cryptographic Module includes a function to verify its own integrity.	
Verdict			

4.3.18 M.2-18 – Recording security functions

Identifier	M.2-18		
Requirement	M2:A6.2-3		
Test Purpose	Check whether the Cryptographic Module has functions to record all security functions in a meaningful and traceable manner.		
Configuration	CONFIG_Common		
Pre-test conditions	• Developer documents are present • Design documents are present		
Step	**Test sequence**	**Expected Results**	**Observations**
1.	Check the vendor documentation for information whether the Cryptographic Module includes a function to record all security functions in a meaningful and traceable manner.	The Cryptographic Module includes a function which records the administration and the exchange of software or keys in a meaningful and traceable manner.	
2.	Check the log files (records) of the Cryptographic module.	The log files record the execution of the security functions in a meaningful and traceable manner.	
Verdict			

4.3.19 M.2-19 – Responsivity to unauthorized access

Identifier	M.2-19		
Requirement	M2:A6.2-4		
Test Purpose	Check whether the Cryptographic Module is capable of cancelling the execution of a function with a meaningful and comprehensible error message in the event of unauthorised access in the module's security functions.		
Configuration	CONFIG_Common		
Pre-test conditions	• Developer documents are present • User manual is present		
Step	**Test sequence**	**Expected Results**	**Observations**

1.	Check the vendor documentation for information whether the Cryptographic Module is capable of cancelling the execution of a function.	The Cryptographic Module is capable of cancelling the execution of functions.	
2.	Check the vendor documentation for information if the capability of cancelling the execution of a function produces a meaningful and comprehensible error message in the event of unauthorised access in the module's security functions.	The cancellation of the execution of a function produces a meaningful and comprehensible error message in the event of unauthorised access in the module's security functions.	
3.	Check the error messages produced during test case M.2-20 due to unauthorized access. Are these error messages meaningful and comprehensible?	All these error messages are meaningful and comprehensible.	
Verdict			

4.3.20 M.2-20 – Configuration of cryptographic functions

Identifier	M.2-20		
Requirement	M2:A6.3-1 M2:A6.3-2		
Test Purpose	Check whether the Cryptographic Module has a central function to configure cryptographic functions. Check whether the configuration is managed by a configuration file and whether this file complies with [RFC5698].		
Configuration	CONFIG_Common		
Pre-test conditions	• User manual is present • Product design documents are present		
Step	**Test sequence**	**Expected Results**	**Observations**
1.	Check the vendor documentation and assess the Crypto Module to identify how the Cryptographic Module realises the configuration of cryptographic functions, especially the algorithms and parameters used for operation.	The Cryptographic Module has a central function to configure cryptographic functions, preferably in a configuration file. or the Crypto Module just supports those algorithms and parameters assessed as suitable for security by the Federal Office for Information Security and the Federal Network Agency (hard-wired) and the Crypto Modules needs to be updated in order to change that.	
2.	If a configuration file is used, check whether this files complies with [DSSC].	The [DSSC] format is used.	
Verdict			

4.3.21 M.2-21 – Verification of certificates based on a standardized protocol

Identifier	M.2-21
Requirement	M2:A5.1-14
Test Purpose	The verification of the validity of the certificate shall occur on the basis of a standardized protocol. (see A5.1-14 in M.2)
Configuration	CONFIG_Common
Pre-test conditions	• User manual is present • Developer documents are present

Step	Test sequence	Expected Results	Observations
1.	Check the user manual and developer documents for information about which protocols for the verification of the validity of certificates are supported.	The list of supported protocols for the verification of the validity of the certificate is given. OCSP is supported.	
2.	Check each other supported verification protocol, if it is standardized.	All other supported protocols for the verification of the validity of the certificate are standardized.	
Verdict			

4.3.22 M.2-22 – Crypto-Module is able to request qualified time stamps

Identifier	M.2-22
Requirement	M2:A5.3-1
Test Purpose	The Cryptographic Module has a function to request a qualified time stamp. The request can be made to a certification service provider or to a device controlled by the Cryptographic Module.
Configuration	CONFIG_Common
Pre-test conditions	• User manual is present • The Cryptographic module may be configured to request a time stamp by a service provider or an internal device

Step	Test sequence	Expected Results	Observations
1.	Check the user manual whether the Cryptographic Module has a function to request a qualified time stamp.	The Cryptographic Module has a function to request a qualified time stamp.	
2.	Request a qualified time stamp using the corresponding interface function a) from a certificated service provider or b) a certificated device controlled by the Cryptographic module.	The request of the qualified time stamp is possible.	
3.	Observe the output of the interface function.	A positive feedback will be received; no error message or error code. The time stamp shall be received.	
4.	Check the time stamp whether it is a qualified one.	The time stamp is a qualified time stamp.	
Verdict			

4.3.23 M.2-23 – Crypto-Module supports RFC 3161 and suitable algorithms

Identifier	M.2-23
Requirement	M2:A5.3-3 M3:A4.7-4
Test Purpose	The Cryptographic Module shall check whether requested time stamp fulfils the requirements and specifications of the time stamp protocol pursuant to

	[RFC3161], [RFC3852] and [ETSI-TSP] and whether the limitations for algorithms and parameters assessed as suitable for security by the Federal Office for Information Security (BSI) and the Federal Network Agency are implemented.
Configuration	CONFIG_Common
Pre-test condition	Install a Time Stamp Service which accepts requests compliant with TSP (RFC 3161) Configure the Crypto Module to use this Time Stamp Service Supply the list of algorithms and parameters assessed as suitable for security by the Federal Office for Information Security (BSI) and the Federal Network Agency

Step	Test sequence	Expected Results	Observations
1.	Configure the Crypto Module according to the guidance; especially the protocol used to access the Time Stamp Service. Check also whether there are guidance hints regarding the configuration of algorithms and other cryptographic parameters.	It is expected that there are at least some hints regarding the configuration of algorithms according to the recommendations of the BNetzA.	
2.	Request the time stamp using the interface function „TimestampRequest“ for each hash algorithm supported by the Cryptographic Module. The requestData contain the corresponding hash-algorithm-identifier.	The request of the qualified time stamp with algorithm-identifier in requestData as parameter is possible. A positive feedback will be received; no error message or error code. The time stamp shall be received for at least one algorithm.	
3.	Request a time stamp using the interface function “TimestampRequest” where the time of the executing the request has been manipulated in such a manner that it differs substantial from the moment of the request.	The crypto module returns an error message indicating that the returned time is incorrect.	
4.	Request a time stamp using the interface function “TimestampRequest” where signature of the timestamp is invalid.	The crypto module returns an error message indicating that the signature of the timestamp is invalid.	
Verdict			

4.3.24 M.2-24 – Time stamps need to bear qualified electronic signature

Identifier	M.2-24
Requirement	M2:A5.3-2
Test Purpose	The Cryptographic Module checks whether requested qualified time stamps include a qualified electronic signature from the time stamp issuer.
Configuration	CONFIG_Common
Pre-test conditions	• Cryptographic Module is configured (if possible) to check whether requested qualified time stamps for re-signing include a qualified electronic signature from the time stamp issuer

Step	Test sequence	Expected Results	Observations
1.	If possible, configure the time stamp service provider or the requesting middleware in such a way that the time stamps will be qualified signed. Otherwise, use a time stamp service provider actually generating qualified signed time stamps.	The test set-up is possible.	
2.	t the Cryptographic Module request a time stamp from the time stamp service provider.	The Cryptographic Module requests the time stamp.	
3.	Observe the output of the Cryptographic Module.	A positive feedback will be received; no error message or error code. The Cryptographic Module accepts the qualified signed qualified time stamp.	
4.	If possible, configure the time stamp service provider or the requesting middleware in such a way that the time stamps will be **not** qualified signed. Otherwise, use a time stamp service provider actually generating signed time stamps but not qualified signed.	The test set-up is possible.	
5.	Let the Cryptographic Module request a time stamp from the time stamp service provider.	The Cryptographic Module requests the time stamp.	
6.	Observe the output of the Cryptographic Module.	A negative feedback will be received; an error message or error code on display or in error log will appear. The Cryptographic Module doesn't accept not qualified signed qualified time stamp.	
7.	If possible, configure the time stamp service provider or the requesting middleware in such a way that the time stamps will be **not** signed.	The test set-up is possible.	

	Otherwise, use a time stamp service provider actually generating not signed time stamps.		
8.	Let the Cryptographic Module request a time stamp from the time stamp service provider.	The Cryptographic Module requests the time stamp.	
9.	Observe the output of the Cryptographic Module.	A negative feedback will be received; an error message or error code on display or in error log will appear. The Cryptographic Module doesn't accept not signed qualified time stamp.	
Verdict			

4.3.25 M.2-25 – Crypto-Module shall verify signatures of received time-stamps

Identifier	M.2-25
Requirement	M2:A5.3-4 M2:A5.3-5
Test Purpose	Check whether the Cryptographic Module verifies the authenticity and integrity of received qualified time stamps immediately upon receipt and prior to further processing including the validation of the certificate chain back to a trustworthy root CA.
Configuration	CONFIG_Common
Pre-test conditions	• Configure Crypto Module to maximum verbose logging

Step	Test sequence	Expected Results	Observations
1.	Request a qualified time stamp using the functions of the Crypto Module.	The Crypto Module performs the request.	
2.	Check log files or other evidences whether the Crypto Module has verified the authenticity and integrity of the received qualified time stamp (the signature).	The Crypto Module has successfully verified the mathematical correctness of the signature.	
3.	Check log files or other evidences whether the Crypto Module has verified the certificate used for signature.	The Crypto Module has verified successfully the signature certificate.	
4.	Check log files or other evidences whether the Crypto Module has verified the CA certificate used to sign the certificate used for signature.	The Crypto Module has verified successfully the CA certificate	
5.	Emulate the check of invalid signatures and certificates.	The Cryptographic module detects and logs the failures.	
Verdict			

4.4 Module 3 – ArchiSig-Module

Pre-supposition:

A product which claims to be conform to the M.3 ArchiSig specification of this TR has to pass

- all test cases in this section and
- all test cases for the interface S.6 specified in section 4.5.6 or prove that it supports functional analogous interfaces.

4.4.1 M.3-01 – ArchiSig-Module should be realised as a separate module

Identifier	M.3-01
Requirement	M3:A3.1-4
Test Purpose	The test shall verify that the ArchiSig-Module runs as an independent application or independent (functionally delimited) part of an application on a trustworthy IT system[6] and is neither a logical nor functional component of upstream IT specialist applications.
Configuration	CONFIG_Common
Pre-test conditions	• User manual is present

Step	Test sequence	Expected Results	Observations
1.	Check TOT and the user manual, whether the ArchiSig-Module is an independent application or independent (functionally delimited) part of an application.	The ArchiSig-Module is an independent application or independent part of an application.	
2.	Check whether the IT system is trustworthy on which the module is implemented. For this purpose the vendor could provide a specially hardened system or could assume a specially hardened system. The test fails, if no settings for the baseline system are assumed or already provided.[7]	There are statements about the trustworthy IT system.	
3.	Check the TOT and/or the user manual, whether the ArchiSig-Module is either a logical or functional component of the upstream IT specialist applications.	The ArchiSig-Module is neither a logical nor functional component of upstream IT specialist applications.	
Verdict			

[6] The term trustworthy has been applied to IT systems that are inherently secure, available and reliable.

[7] For example, if the vendor just states that the product runs on the platform XYZ, the test fails.
If the vendor states that the products runs on the platform XYZ and the security white paper of the vendor of this platform have to be considers, the test passes.

4.4.2 M.3-02 – Using interface S.3 is possible

Pre-supposition:

A product which claims to comply with the interfaces specification S.3, of this TR or part of it has to pass the following test case or part of it or prove that they support functional analogous interfaces.

Identifier	M.3-02
Requirement	M3:A3.2-2
Test Purpose	The test shall verify that the ArchiSig module is able to access the other modules of the middleware via dedicated interfaces as described in the annexes TR-ESOR-M.2 and TR-ESOR-S of this technical guideline.
Configuration	CONFIG_ArchiSig
Pre-test conditions	• The middleware documentation is available

Step	**Test sequence**	**Expected Results**	**Observations**
1.	Check whether the ArchiSig documentation contains the description of how to connect to the interface S.3.	The interface is described in the documentation.	
2.	Check whether it is possible for the ArchiSig module to communicate with the Crypto Module via the S.3 interface.	Communication is possible.	
Verdict			

4.4.3 M.3-03 – ArchiSig-Module implements specified functions

Identifier	M.3-03
Requirement	M3:A4.0-1
Test Purpose	The test shall verify that an ArchiSig-Module provides at least the following functions: • Archive Submission • Generation of an AOID • Performing canonicalisation • Generating hash values (using a Crypto-Module) • Generating an initial time stamp (using a Crypto-Module) • Passing archive objects to the storage • Renewal of Archive Time Stamps • Renewal of hash trees • Generating an Evidence Record (ER) for a specified archive object
Configuration	CONFIG_Common
Pre-test conditions	• User manual is present

Step	Test sequence	Expected Results	Observations
1.	Check whether a “Archive Submission” function exists.	Yes, such a function exists.	
2.	Check whether a function for the generation of AOIDs exists or the guidance states that this function shall be provided by other modules like the storage.	Yes, such a function exists or the feature is declares to be done by another module.	
3.	Check whether a function for XML canonicalisation exists. Note: For products which supports the storage (processing) of BIN data only this step may be passed as fulfilled.	Yes, ArchiSig ensures that all XML objects are canonicalised before hashed.	
4.	Check whether ArchiSig is able to generate hash values, by using a Crypto Module.	Yes, ArchiSig is able to calculate hash values by using a Crypto Module.	
5.	Check whether ArchiSig is able to generate initial Archive Time Stamps (ATS), by using a Crypto Module.	Yes, ArchiSig is able to calculate ATS by using a Crypto Module.	
6.	Check whether ArchiSig passes the archive objects to the storage system.	Yes, ArchiSig passes all objects to the storage after hashing.	

7.	Check whether ArchiSig renews the Archive Time Stamps.	Yes, ArchiSig is able to calculate and renew ATS by using a Crypto Module.	
8.	Check whether ArchiSig is able to renew the hash trees. For this purpose, ArchiSig must be able to read the archive objects from the storage.	Yes, ArchiSig is able to renew hash trees. For this purpose it reads the archive objects from the storage.	
9.	Check whether ArchiSig is able to generate an ERS record conform to RFC 4998[8] or RFC 6283 for a specific archive object.	Yes, ArchiSig is able to generate an ERS record conform to RFC 4998 or RFC 6283 for every archive object.	
Verdict			

[8] **[RFC4998]** must be supported, **[RFC6283]** can be supported.

4.4.4 M.3-04 – Creation of initial archive time stamps

Identifier	M.3-04
Requirement	M3:A4.5-4
Test Purpose	The test should verify that the creation of the Initial Archive Time Stamp is automated and take place according to configurable rules reliably stored in the ArchiSig-Module.
Configuration	CONFIG_Common
Pre-test conditions	• User has administrator rights on the system • If required, perform identification and authentication • At least one archive object is already archived

Step	Test sequence	Expected Results	Observations
1.	Check the ArchiSig-Module, whether there are configurable rules for the creation of Initial Archive Time Stamps.	There are configurable rules for the creation of Initial Archive Time Stamps.	
2.	Configure the ArchiSig-Module in such a way that every 10 minutes (or another short time period) a new Archive Time Stamp will be created.	Configuration is possible.	
3.	Request every 10 minutes (or the configured period of time) a new ER of an already archived object (3 or 4 times).	ER can be retrieved.	
4.	Check the last Initial Archive Time Stamp.	The check is performed successfully.	
Verdict			

4.4.5 M.3-05 – AOID shall be unique

Identifier	M.3-05		
Requirement	M3:A4.2-2		
Test Purpose	The test shall verify that the generation of an AOID shall be unique (collision free).		
Configuration	CONFIG_Common		
Pre-test conditions	• User has administrator rights on the system • If required, perform identification and authentication • Test cases S.4-14 and S.4-19 were performed successful and the AOIDs are known		
Step	**Test sequence**	**Expected Results**	**Observations**
1.	Compare the known AOIDs.	No two AOIDs are equal.	
Verdict			

4.4.6 M.3-06 – ArchiSig-Module creates Evidence Records according to RFC4998 or RFC6283

Identifier	M.3-06
Requirement	MD:A5.1-22 MD:A5.1-23 M3:A3.1-1 M3:A4.10-1
Test Purpose	Check whether the Middleware is able to provide technical evidence for the authenticity and unsophisticatedness of the archival information packages upon request as well as all electronic Evidence Records needed for this purpose. The function shall calculate all Evidence Records (ERs) pursuant to the ERS standard for an Archival Information Package identified uniquely by the AOID, and the result shall be returned in an allowed format [RFC4998] / [RFC6283][9] to the application or module making the request.
Configuration	CONFIG_Common
Pre-test conditions	• user manual is present • Tester has read/write permissions on the Middleware • If required, perform identification and authentication

Step	Test sequence	Expected Results	Observations
1.	Check the user manual if an Evidence Record is calculated pursuant to the ERS standard for an Archival Information Package identified uniquely by the AOID.	The Evidence Record is calculated pursuant to the ERS standard [RFC4998] or [RFC6283].	
2.	Using the interface function “ArchiveEvidenceRequest“ and an existing AOID request all Evidence Records identified uniquely by the AOID.	The call of the function with this AOID as parameter is possible.	
3.	Observe the output of the interface function “ArchiveEvidenceResponse”.	A positive feedback will be received; no error message or error code. All Evidence Records idenified uniquely by the AOID will be received.	
4.	Check whether each Evidence Record contains the hash values according to the version of the archive object.	Each Evidence Record contains the hash values according the version.	
5.	Check whether each Evidence Record contains an Archive Time Stamp Sequence which demonstrates the integrity of the archive object.	Each Evidence Record contains such a sequence.	
6.	Check whether the time stamps of the Archive Time Stamp Sequence are qualified time stamps and contain a	All time stamps are qualified time stamps, i.e. time stamps completed by a qualified signature.	

[9] **[RFC4998]** must be supported, **[RFC6283]** can be supported,

	qualified electronic signature, which demonstrates the integrity and possibly the authenticity of the archive object.		
7.	Check whether the Evidence Record has an allowed format [RFC4998] / [RFC6283].	The Evidence Record has an allowed format [RFC4998] / [RFC6283].	
Verdict			

4.4.7 M.3-07 – ArchiSig-Module should not implement cryptographic functions

Identifier	M.3-07
Requirement	M3:A3.1-5 M3:A4.4-3
Test Purpose	The test shall verify that the ArchiSig-Module itself has not implemented cryptographic functions for the protection of the authenticity or verification of the integrity and authenticity with the exception of the canonicalisation functions and the functions for generation of Merkle hash trees.
Configuration	CONFIG_Common
Pre-test conditions	• Disconnect the Crypto Module from the ArchiSig Module • User manual is present • User has administrator rights on the system • If required, perform identification and authentication

Step	Test sequence	Expected Results	Observations
1.	Check, whether ArchiSig could be configured in such a way that no Crypto Module needs to be used.	The TOT may or may not have such a configuration option. If it does not, this test case is finished and considered to be passed. If it does, the security guidance of the vendor clearly states that this configuration is not recommended..	
2.	Call the “ArchiveSubmissionRequest” function of ArchiSafe using XAIP_OK or BIN as parameter. If required, perform identification and authentication.	The call of the function with this XAIP / BIN as parameter is possible.	
3.	Observe the output of the interface function “ArchiveSubmissionResponse”.	A negative feedback will be received; an error message or error code should show that the signature cannot be verified because a hash value for the XAIP / BIN couldn't be calculated.	
Verdict			

4.4.8 M.3-08 – ArchiSig-Module should be thread safe

Identifier	M.3-08
Requirement	MD:A7.3-9 MD:A7.3-10 MD:A7.3-11
Test Purpose	The ArchiSig-Module should be able to work parallel in multiple entities, in particular with regard to the case when all archival information packages present in the ECM/long-term storage have to be re-signed and/or to be re-hashed.
Configuration	CONFIG_Common If possible, configure ArchiSig to work parallel in multiple entities on one computer; consult the guidance for that purpose.
Pre-test conditions	• Test case S.4-14 was performed successful and the AOID is noted • User has administrator rights on the system • User manual is present • If required, perform identification and authentication • Ensure that there are a lot (several thousand) archive objects in the archive

Step	Test sequence	Expected Results	Observations
1.	Start a complete resigning of the archival information packages.	The resigning of the archival information packages starts.	
2.	his must be done during the resigning Request some archival information package from the TOT using the interface function "ArchiveRetrievalRequest" and the noted AOID from the test case S.4-14.	The call of the function with this AOID as parameter is possible. The results were received in an acceptable amount of time.	
3.	his must be done during the resigning Submit some archival information package to the TOT using the interface function "ArchiveSubmissionRequest".	The call of the function is possible. The results (the AOID) were received in an acceptable amount of time.	
4.	Start a complete rehashing of the archival information packages.	The rehashing of the archival information packages starts.	
5.	This must be done during the rehashing Request some archival information package from the TOT using the interface function "ArchiveRetrievalRequest" and the noted AOID from the test case S.4-14.	The call of the function with this AOID as parameter is possible. The results were received in an acceptable amount of time.	

6.	This must be done during the rehashing Submit some archival information package to the TOT using the interface function "ArchiveSubmissionRequest".	The call of the function is possible. The results (the AOID) were received in an acceptable amount of time (<= 2 Min).	
Verdict			

4.4.9 M.3-09 – Instances of ArchiSig-Module should be deployable on different machines

Identifier	M.3-09		
Requirement	AS:A5.2-1 AS:A5.3-1 AS:A5.6-1 M3:A3.2-2 MD:A7.3-10		
Test Purpose	The individual entities of ArchiSig should be able to run on different machines.		
Configuration	CONFIG_Common		
Pre-test conditions	If required, perform identification and authentication		
Step	**Test sequence**	**Expected Results**	**Observations**
1.	Perform test case M.3-01	This demonstrates that multiple entities on one computer work.	
2.	Configure ArchiSig in such a way that the multiple entities are running on different computers. Consulting the guidance for that purpose.	That should be possible.	
3.	Perform test case M.3-01 again.	This demonstrates that multiple entities on different computers work.	
Verdict			

4.4.10 M.3-10 – ArchiSig-Module uses a secure storage for time stamps and AOIDs

Identifier	M.3-10
Requirement	M3:A3.1-6 M3:A4.4-4
Test Purpose	The test shall verify that the calculated hash value H_{XAIP} or H_{BIN} and the AOID and, if applicable, version ID will be stored and preserved in secure data storage that is part of or allocated to the ArchiSig-Module in such a way that a hash value corresponding to an AOID and, if applicable, version ID can be identified with absolute certainty at any time.
Configuration	CONFIG_Common
Pre-test conditions	• XAIP_OK was archived successful and updated several times to obtain several versions • XAIP(BIN) was archived successful and updated several times to obtain several versions • The versionIDs were noted • If required, establish a session with the TOT in order to perform the following tests • If required, perform identification and authentication

Step	Test sequence	Expected Results	Observations
1.	Use the interface function “ArchiveRetrievalRequest” and the AOID from the archived XAIP_OK or XAIP(BIN) .	The call of the function with this AOID as parameter is possible and the latest version of XAIP_OK or XAIP(BIN) will be received.	
2.	Use the interface function “ArchiveRetrievalRequest”, the AOID from the archived XAIP_OK or XAIP(BIN) and an older versionID.	The call of the function with this AOID and versionID as parameters is possible and the appropriate version of XAIP_OK or XAIP(BIN) will be received.	
3.	Use the interface function “ArchiveEvidenceRequest”, the AOID from the archived XAIP_OK or XAIP(BIN) and an older versionID.	The call of the function with this AOID and versionID as parameters is possible and • the appropriate Evidence Records of XAIP_OK or XAIP(BIN) will be received. • The retrieved Evidence Records could be positively verified by an appropriate tool.	
4.	Use the interface function “ArchiveRetrievalRequest” and the AOID from the archived XAIP_OK or XAIP(BIN) .	The call of the function with this AOID as parameter is possible and • the latest version of XAIP_OK or BIN embedded in an XAIP (XAIP(BIN)) will be received	
5.	Use the interface function “ArchiveRetrievalRequest”, the AOID from the archived XAIP_OK or XAIP(BIN)	The call of the function with this AOID and versionID as parameters is possible and	

	and an older versionID.	• the appropriate version of XAIP_OK or XAIP(BIN) embedded in an XAIP will be received	
6.	Use the interface function “ArchiveEvidenceRequest”, the AOID from the archived XAIP_OK or XAIP(BIN) and an older versionID.	The call of the function with this AOID and versionID as parameters is possible and • the appropriate Evidence Records of XAIP_OK or XAIP(BIN) will be received. • The retrieved Evidence Records could be positively verified by a appropriate tool.	
7.	Compare the hash values of the Evidence Records.	The hash values of the two quantities of Evidence Records are not equal. This demonstrates that per archive object and also per version of archive objectunique hash values will be generated.	
Verdict			

4.4.11 M.3-11 – Canonicalisation of XML is performed prior to hashing and noted in XAIP

Identifier	M.3-11
Requirement	M3:A4.3-1 M3:A4.3-2
Test Purpose	The test shall verify that the algorithm used for the canonicalisation is entered into the corresponding field of the Package Headers of the XAIP before the canonicalisation and hash value calculation. Note: In the case the product supports submission of BIN data only, the test case may passed as fulfilled.
Configuration	CONFIG_Common
Pre-test conditions	• Developer documents are present

Step	Test sequence	Expected Results	Observations
1.	Prepare an XAIP_OK in such a way that it is not canonicalised (e.g. entering some blanks between tags) Check that no AOID and no canonicalisation algorithm is stated in the XAIP.	---	
2.	Submit this special XAIP to the archive using the respective S.4 function.	This works without error.	
3.	Retrieve this special XAIP using the respective S.4 function.	This works. The XAIP is retrieved.	
4.	Compare the retrieved XAIP and the original XAIP.	The retrieved XAIP is canonicalised and the AOID and the canonicalisation algorithm is stated in the XAIP	
5.	Retrieve the ERs for the special XAIP using the respective S.4 function. Calculate the hash values for the special XAIP and the XAIP retrieved in step 3 manually (see annex TR-ESOR-M.3 chapter 2.4.1 for details).	The ERs can be retrieved. The hash value used in the ERs matches to the canonicalised XAIP containing the AOID and the canonicalisation algorithm.	

Verdict

4.4.12 M.3-12 – Hashing of relevant parts is performed with suitable algorithms

Identifier	M.3-12
Requirement	M3:A4.4-1 M3:A4.4-2
Test Purpose	The test shall verify that the calculation of the hash value for the relevant parts of the Archival Information Package is based on algorithms and parameters which are capable to protect the security for long-terms.
Configuration	Config_COMMON
Pre-test conditions	• User manual is present

Step	**Test sequence**	**Expected Results**	**Observations**
1.	Check the user manual, whether the calculation of the hash value, done by the Cryptographic Module in order of the ArchiSafe-module, is done on the basis of suitable algorithms and parameters as recommended by the BNetzA.	The calculatation of the hash value is done on the basis of at least one of the recommended algorithms and parameters according to BnetzA, which can be configured in the ArchiSigModule or Cryptographic Module.	
Verdict			

4.4.13 M.3-13 – ArchiSig-Module supports time stamp renewal and hash tree renewal

Identifier	M.3-13
Requirement	MD:A5.1-6 MD:A5.1-7
Test Purpose	The test shall verify that pursuant to §17 SigV the signed data can be re-signed and re-hashed.
Configuration	CONFIG_Common
Pre-test conditions	• Test user has administrative rights on the system • There are XAIPs or BINs stored in ECM/long-term storage and their AOID's are known • If required, perform identification and authentication

Step	Test sequence	Expected Results	Observations
1.	Use several interface functions “ArchiveEvidenceRequest” with the known AOIDs. If required, perform identification and authentication.	Several calls of the function with the AOIDs as parameters are possible. Appropriate Evidence Records will be received.	
2.	Start the the re-sign (time stamp renewal) process based on interfaces provided by the ArchiSig module.	The initiation of the re-sign process is possible. No error is indicated.	
3.	Check log for information about the re-sign process.	No error messages or error codes for the re-signing are in the log.	
4.	Use several interface functions “ArchiveEvidenceRequest” with the known AOIDs.	Appropriate Evidence Records will be received.	
5.	Compare the new Evidence Records with the old Evidence Records of the XAIPs or BINs from step 1.	The new and the old Evidence Records are not equal. The new Evidence Records base on the new signature algorithms.	
6.	Change old hash-algorithm against new one.	The change of Hash-Algorithm is possible.	
7.	Initiate re-hash (hash tree renewal) process.	The initiation of the re-hash process is possible.	
8.	Check log for information about the re-hash process.	No error messages or error codes for the re-hashing are in the log.	
9.	Start the the re-sign (time stamp renewal) process based on interfaces provided by the ArchiSig module.	The initiation of the re-sign process is possible. No error is indicated.	
10.	Check log for information about the re-sign process.	No error messages or error codes for the re-signing are in the log.	

11.	Use several interface functions “ArchiveEvidenceRequest” with the known AOIDs.	Appropriate Evidence Records will be received.	
12.	Compare the new Evidence Records with the old Evidence Records of the XAIPs or BINs from step 1 and step 4.	The new and the old Evidence Records from step 1, 4 and 12 are not equal. The new Evidence Records base on the new hash and signature algorithms.	
13.	Use several interface functions “ArchiveRetrievalRequest” with the known AOIDs.	The XAIP's are retrieved from the storage.	
14.	Check the credential section of the XAIPs.	The respective “old” Evidence Records with old hash value are included in the credential section.	
Verdict			

4.4.14 M.3-14 – Time stamp renewal

Identifier	M.3-14
Requirement	M3:A4.5-1 M3:A4.5-2 M3:A4.7-1
Test Purpose	The test shall verify that when the function for renewal of the Archive Time Stamp is requested, the latest Archive Time Stamp will be renewed .
Configuration	CONFIG_Common
Pre-test conditions	• User manual is present • User has administrator rights on the system • If required, perform identification and authentication • There are already archived Archival Information Packages without Archive Time Stamp in the ECM/long-term storage

Step	**Test sequence**	**Expected Results**	**Observations**
1.	Use the function for renewal of the Archive Time Stamp.	The renewal of the latest Archive Time Stamps is done.	
2.	Request the ERs for the archive object archived or updated at the very last.	The ERs must contain the hash value of the archive object and an initial time stamp. The time stamp should show the time of calling the function in step 1 or an earlier time.	
3.	Disconnect the Crypto Module from the ArchiSig Module and perform this test case again.	The calculation of the initial Archive Time Stamp (the hash value) is not possible because ArchiSig itself does not have this functionality.	
Verdict			

4.4.15 M.3-15 – ArchiSig-Module shall verify requested time stamps

Identifier	M.3-15
Requirement	M3:A4.5-3 M3:A4.7-5 M3:A4.8-2 M3:A4.8-5
Test Purpose	The ArchiSig-Module shall in case of generating new time stamps ensure that the time stamp contains all information required for validation of the time stamp, including the qualified electronic signatures contained therein. In case of renewal of the hash trees the time stamp shall contain all information required for validation of the time stamp, including the qualified electronic signatures contained therein. The concluding Archive Time Stamp of the hash trees to be renewed will be re-verified for integrity and authenticity before these Archive Time Stamps are transferred into a new hash tree or included there. To do so, the signature of this Archive Time Stamp and the associated certificate chain will be re-verified with the help of the functions of the TR-ESOR-M.2 Cryptographic Module. An inclusion of this Archive Time Stamp in the new hash tree only takes place if this verification has had a positive result.
Configuration	CONFIG_Common
Pre-test conditions	• ECM/long-term storage contains already some objects and AOIDs are known • Tester emulate a TR-ESOR M.2 Cryptographic Module • Test case M.3-16 was performed successfully • Some archive objects are already archived

Step	Test sequence	Expected Results	Observations
1.	Ensure that ArchiSig creates a new Archive Time Stamp (e.g. by using a Crypto Module).	ATS is generated.	
2.	Request an Evidence Records for one known AOID.	Requesting of an Evidence Record was performed successfully.	
3.	Check the Evidence Record for information about time stamps and verifications (OCSP Responses, CRL-Reports) of signatures of time stamps.	The information about the time stamps, its signatures and the verification information of the signatures are present and show all information required for validation of the time stamp up to the certificate of a trustworthy root CA.	
4.	Start the hash-tree renewal process.	The hash-tree renewal process was started successfully.	
5.	Observe the requests of the ArchiSig module to the Cryptographic Module.	ArchiSig will request verification of the very last Archive Time Stamp signature.	
6.	Emulation: the Cryptographic Module send negative	Sending of negative response was performed successfully.	

	response.		
7.	Check the log files of the ArchiSig-Module or observe otherwise the reaction of ArchiSig.	ArchiSig should at least mention the failed verification of the qualified time stamp. The ArchiSig module must stop the has tree renewal and log an exception.	
8.	Request an Evidence Record for one known AOID.	Requesting of an Evidence Record was performed successfully.	
9.	Check the Evidence Records by an appropriate tool for information about the Archive Time Stamp and signature check (OCSP Responses, CRL-Reports).	The check of the tool shows that the ERs resp. the time stamp chain is not integer.	
10.	Start the hash-tree renewal process manually or wait the preconfigured period of time till automatic renewal process.	The hash-tree renewal process was started successfully.	
11.	Observe the requests of the ArchiSig module to the Cryptographic Module.	ArchiSig will request verification of the very last Archive Time Stamp signature.	
12.	Emulation: the Cryptographic Module sends positive response.	Sending of positive response was performed successfully.	
13.	Check the log files of the ArchiSig-Module or observe otherwise the reaction of ArchiSig.	ArchiSig should continue and finish the hash tree renewal.	
14.	Request an Evidence Record for one known AOID.	Requesting of an Evidence Record was performed successfully.	
15.	Check the Evidence Records by an appropriate tool for information about the archive time stamp and signature check of steps 11./12./13. (OCSP Responses, CRL-Reports)	The check of the tool shows that the ERs resp. the time stamp chain for the steps 11./12../13. is integer and for the steps 5./6./7. is not integer.	
Verdict			

4.4.16 M.3-16 – Time stamps shall be verified prior to renewal

Identifier	M.3-16
Requirement	M3:A4.7-2 M3:A4.7-3
Test Purpose	Check, whether a complete Archive Time Stamp renewal verifies the integrity and authenticity of the Archive Time Stamps to be renewed and whether the hash values of these Archive Time Stamps are included in the new Archive Time Stamp.
Configuration	CONFIG_Common
Pre-test conditions	• Submit several archive objects to the storage and configure the automatic Archive Time Stamping in such a way, that several Archive Time Stamps will be generated in parallel and they are not "covered" by a superior Archive Time Stamp • If required, perform identification and authentication

Step	Test sequence	Expected Results	Observations
1.	Request the ERs of these archive objects which are covered by the mentioned parallel Archive Time Stamps.	The hash value of each of the parallel Archive Time Stamps is documented in one ERS.	
2.	Start the complete Archive Time Stamp renewal process.	The complete Archive Time Stamp renewal process was started successfully.	
3.	Observe the requests of the ArchiSig module to the Cryptographic Module.	ArchiSig will request verification of the very last Archive Time Stamp signature.	
4.	Emulation: the Cryptographic Module sends a negative response.	Sending of negative response was performed successfully.	
5.	Check the log files of the ArchiSig-Module or observe otherwise the reaction of ArchiSig.	ArchiSig shall mention the failed verification of the qualified time stamp and stop the complete Archive Time Stamp.	
6.	Request an Evidence Records for one known AOID.	Requesting of an Evidence Records was performed successfully.	
7.	Check the Evidence Records (ERs) for information about the Archive Time Stamp and signature check of steps 3./4./5. (OCSP Responses, CRL-Reports).	The ERs should contain no new Archive Time Stamp.	
8.	Start the complete Archive Time Stamp renewal process.	The complete Archive Time Stamp renewal process was started successfully.	
9.	Observe the requests of the ArchiSig module to the Cryptographic Module.	ArchiSig will request verification of the very last Archive Time Stamp signature.	

10.	Emulation: the Cryptographic Module send positive response.	Sending of positive response was performed successfully.	
11.	Check the log files of the ArchiSig-Module or observe otherwise the reaction of ArchiSig.	ArchiSig should continue and finish the complete Archive Time Stamp renewal.	
12.	Request an Evidence Records for one known AOID	Requesting of an Evidence Records was performed successfully.	
13.	Check the Evidence Records for information about the Archive Time Stamp and signature check of steps 9./10./11. (OCSP Responses, CRL-Reports) and the hash algorithm used for this time stamp.	The ERs should contain the new archive time stamp. All the hash values of the parallel Archive Time Stamps are covered by the new Archive Time Stamp.	
Verdict			

4.4.17 M.3-17 – Time stamp renewal can only be requested by authorised users through administrative interfaces

Identifier	M.3-17
Requirement	M3:A4.7-6
Test Purpose	The test shall verify that the function “Renewal of Archive Time Stamp” can – beside the automated function – only be requested manually by authorised users through administrative interfaces and will be logged.
Configuration	CONFIG_Common
Pre-test conditions	• If required, perform identification and authentication

Step	Test sequence	Expected Results	Observations
1.	Check whether there is a function “Renewal of Archive Time Stamp” for manual start of the renewal process at all.	There may be a function or not. If not, the remaining test steps do not need to be performed, and the test is considered to be passed.	
2.	Use the function “Renewal of Archive Time Stamp” with an user who has administrator rights on the system.	A positive feedback will be received; no error message or error code.	
3.	Check the log files of the ArchiSig-Module, if there is information about the renewal of Archive Time Stamps.	There is information about the renewal of Archive Time Stamps.	
4.	Use the function “Renewal of Archive Time Stamp” with a user who has no administrator rights on the system.	A call of the function is not possible and a clear and understandable error message or error code will be received.	
5.	Check the log files of the ArchiSig-Module, if there is information about the try of renewing Archive Time Stamps.	There is no information that the function was performed successfully, but there shall be information about the failed request.	
Verdict			

4.4.18 M.3-18 – Hash tree renewal can only be requested through administrative interface

Identifier	M.3-18
Requirement	M3:A4.8-1 M3:A4.8-3 M3:A4.8-4
Test Purpose	The test shall check whether the function “Renewal of Hash Tree” calculates new hash values on the basis of configured hash algorithm for all archival information packages stored in the ECM/long-term storage that have been registered by the TR-ESOR-Middleware as well as the Archive Time Stamp sequences stored in the data storage of the ArchiSig-Module.
Configuration	CONFIG_Common
Pre-test conditions	• Test user has administrative rights on the system • There are XAIPs / BINs, registered by the TR-ESOR-Middleware, stored in ECM/long-term storage • There are XAIPs / BINs stored in ECM/long-term storage, which are not registered by the TR-ESOR-Middleware • If required, perform identification and authentication • Perform test case S.3-04 also together with this test case

Step	Test sequence	Expected Results	Observations
1.	Change the hash algorithm configuration of the Crypto Module so that another algorithm will be used since now.	Configuration is possible (even if the complete Crypto Module must be replaced for that purpose).	
2.	Configure the storage in such a way that the access to objects can be traced (e.g. activate detailed logging).	Tracing of every object access is activated.	
3.	Use the function “Renewal of Archive Time Stamp” (with an administrative user).	A positive feedback will be received; no error message or error code.	
4.	Check the ECM/long-term storage, whether objects, which are not registered by the TR-ESOR-Middleware, can be accessed by the middleware.	The middleware should not have accessed these objects.	
5.	Check the ECM/long-term storage, if XAIPs / BINs, which are registered by the TR-ESOR-Middleware, get a new hash value.	The middleware should have accessed these objects.	
6.	Request the ERs for all these objects.	It can be demonstrated that every XAIP / BIN got a new hash value with the new configured algorithm and that the “old” Archive Time Stamp sequences are also covered by the hash tree renewal (see M.3 sec. 2.4.4).	

Verdict

4.4.19 M.3-19 – Authenticity and integrity of ArchiSig-Module needs to be guaranteed

Identifier	M.3-19
Requirement	M3:A5.1-3
Test Purpose	Check whether the authenticity and integrity of the installed ArchiSig-Module is guaranteed during operation.
Configuration	CONFIG_Common
Pre-test conditions	• User manual is present • Development and design documents are present

Step	Test sequence	Expected Results	Observations
1.	Check the user manual whether there are statements how to ensure the authenticity and integrity of the installed ArchiSig-Module during operation.	The guidance contains such statements and the statements are clear and intelligible.	
2.	Check whether the ArchiSig-Module is a signed software module.	The ArchiSig-Module is signed or otherwise integrity-proteced (e.g. hardware sealed).	
3.	Check the user manual whether the ArchiSig-Module includes a function to verify its own integrity as self-defence against manipulation.	The ArchiSig-Module includes a function to verify its own integrity as self-defence against manipulation.	
Verdict			

4.4.20 M.3-20 – ArchiSig-Module should be able to maintain parallel hash-trees

Identifier	M.3-20
Requirement	M3:A5.2-2
Test Purpose	Check whether the ArchiSig-Module returns several reduced Evidence Records when parallel hash-trees are managed.
Configuration	CONFIG_Common
Pre-test conditions	• Configure ArchiSig in such a way that at least two parallel hash-trees are managed • Archive (submit) several archive objects to build up the trees • Ensure that at least one initial Archive Time Stamp is created to build up the trees

Step	Test sequence	Expected Results	Observations
1.	Request the ERs of archive objects submitted to the archive.	The ERs for these archive objects can be retrieved.	
2.	Check the ERs whether there are reduced Evidence Records for every managed hash-tree included.	For every managed hash-tree a separate Evidence Record proofs the integrity of the archive object.	

Verdict

4.4.21 M.3-21 – Resigning-procedure is efficient and compatible with ERS

Identifier	M.3-21		
Requirement	MD:A5.1-8		
Test Purpose	The test shall verify that the solution for re-signing shall be efficient and compatible with the „Evidence Record Syntax“.		
Configuration	CONFIG_Common		
Pre-test conditions	• User manual and developer documents are present		
Step	**Test sequence**	**Expected Results**	**Observations**
1.	Check user manual for re-signing solution.	The solution for re-signing is efficient while it preserves the marketability of the protected documents. Especially the algorithm used has a much better runtime cost model than O(n) when n is the number of documents in the storage.[10]	
2.	Check user manual for re-signing solution.	The solution for re-signing is compatible with the „Evidence Record Syntax“ according to [RFC4998] or [RFC6283].	
Verdict			

[10] http://en.wikipedia.org/wiki/Big_O_notation#Use_in_computer_science

4.4.22 M.3-22 – Deletion of an archive object shall not impair the conclusiveness of others

Identifier	M.3-22
Requirement	MD:A5.1-28
Test Purpose	The test shall verify that the conclusiveness of the remaining documents in the ECM storage is not affected by the deletion of individual XAIPs or BINs.
Configuration	CONFIG_ArchiSafe
Pre-test conditions	• Tester has read/write permissions on the middleware • If required, perform identification and authentication

Step	Test sequence	Expected Results	Observations
1.	Store an XAIP_OK_SIG or BIN using the interface function “ArchiveSubmissionRequest”.	The call of the function with this XAIP / BIN as a parameter is possible.	
2.	Observe the output of the interface function “ArchiveSubmissionResponse”.	A positive feedback is received. No error message or error code occurs. An AOID is assigned.	
3.	Store another XAIP_OK_SIG or BIN using the interface function “ArchiveSubmissionRequest”.	The call of the function with this XAIP / BIN as a parameter is possible.	
4.	Observe the output of the interface function “ArchiveSubmissionResponse”.	A positive feedback is received. No error message or error code occurs. Another AOID is assigned.	
5.	Perform an “ArchiveEvidenceRequest” with the AOID received in step 2.	The function call is possible.	
6.	Observe the output of the interface function “ArchiveEvidenceResponse”.	An Evidence Record for the XAIP / BIN that has been stored in step 1 is received.	
7.	Perform an “ArchiveEvidenceRequest” with the AOID received in step 4.	The function call is possible.	
8.	Observe the output of the interface function “ArchiveEvidenceResponse”.	An archive Evidence Record for the XAIP / BIN that has been stored in step 3 is received.	
9.	Using the interface function “ArchiveDeletionRequest” and the AOID from step 2, delete the XAIP_OK_SIG or BIN.	The call of the function with this AOID as a parameter is possible.	
10.	Observe the output of the interface function “ArchiveDeletionResponse”.	A positive feedback is received. No error message or error code occurs. The XAIP / BIN is deleted.	

11.	Perform an “ArchiveEvidenceRequest” with the AOID received in step 4.	The function call is possible.	
12.	Observe the output of the interface function “ArchiveEvidenceResponse”.	An Evidence Record for the XAIP / BIN that has been stored in step 3 is received.	
13.	Compare the two Evidence Records of the XAIP / BIN that was stored in step 3.	The Evidence Records are equal. It may be possible that in the meantime an automated time stamp renewal of a hash tree renewal occurred. This would be reflected in the ERS.	
Verdict			

4.5 Interface functions

Note: The following test specifications are based on the recommended reference architecture in chapter 7.1 of the main document of this technical guideline. Thus, in the following differences between expected and observed test results should be carefully interpreted by the testers respecting the fact that actual implementations of components and / or modules of the middleware may deviate from the recommended reference architecture. This may result also in different characteristics of implemented and provided interfaces. It is worth noting therefore, that testing the conformity level 1 the referred interfaces are required in a logical functional manner only and not in a technical interoperable characteristic.

4.5.1 Interface S.1

The primary purpose of the TR-ESOR-S.1 interface between the ArchiSafe module and the Cryptographic module is the verification and creation of electronic signatures that were or should be attached to electronic data to be archived (XAIP or BIN documents).

Pre-supposition:

A product which claims to functionally comply with the Interface S.1 specification of this TR has to pass all test cases in this section or prove that it supports functional analogous interfaces.

4.5.1.1 VerifyRequest

4.5.1.1.1 S.1.1-01 VerifyRequest – Verification of signature includes certificate path validation and Evidence Records

Identifier	S.1.1-01
Requirement	M2:A5.1-8 M2:A5.1-9
Test Purpose	The function is able to verify whether the user certificate used to generate the signature was valid at the time the signature was generated (see Chapter 5.1.3). Validity verification shall be complete, i.e. it includes the entire certificate chain back to a trustworthy root certificate. The Cryptographic Module shall be able to verify advanced and qualified electronic signatures. Qualified time stamps with (qualified) electronic signatures as well as Evidence Records shall be verifiable, i.e. the validity of the time stamp signature at the time of time stamp generation must be verified.
Configuration	CONFIG_Common
Pre-test conditions	• An XAIP_OK_Sig_Q / BIN_OK_Sig_Q is present. XAIP_OK_Sig_Q / BIN_OK_Sig_Q is a XAIP_OK_SIG / BINg with qualified electronic signature • An XAIP_OK_Sig_A / BIN_OK_Sig_A is present. XAIP_OK_Sig_A / BIN_OK_Sig_A is a XAIP_OK_SIG / BIN with advanced electronic signature • An XAIP_OK_Sig_Q_ERS is present. XAIP_OK_Sig_Q_ERS is a XAIP_OK_SIG_OK_ER with qualified electronic signature and at least one evidence record • An XAIP_OK_Sig_A_ERS is present. XAIP_OK_Sig_A_ERS is a XAIP_OK_SIG_OK_ER with advanced electronic signature and at least one evidence record

	• An XAIP_NOK_Sig_Q / BIN_NOK_Sig_Q is present. XAIP_NOK_Sig_Q / BIN_NOK_Sig_Q is a XAIP_NOK_SIG / BIN_NOK_SIG with qualified electronic signature • An XAIP_NOK_Sig_A / BIN_NOK_Sig_A is present. XAIP_NOK_Sig_A / BIN_NOK_Sig_A is a XAIP_NOK_SIG / BIN_NOK_SIG with advanced electronic signature • An XAIP_NOK_Sig_Q_ERS is present. XAIP_NOK_Sig_Q_ERS is a XAIP_NOK_SIG_OK_ER with qualified electronic signature and at least one evidence record • An XAIP_NOK_Sig_A_ERS is present. XAIP_NOK_Sig_A_ERS is a XAIP_NOK_SIG_OK_ER with advanced electronic signature and at least one evidence record • An XAIP_NOK_ERS is present. XAIP_NOK_ERS is a XAIP_NOK_ER with qualified electronic signature and at least one evidence record • A DXAIP_OK_SIG is present. DXAIP_OK_SIG is a DXAIP_OK_SIG with qualified or advanced electronic signature referenced to an XAIP_OK_SIG • A DXAIP_NOK_SIG is present. DXAIP_NOK_SIG is a DXAIP_NOK_SIG with qualified or advanced electronic signature referenced to an XAIP_OK_SIG or XAIP_NOK_SIG • developer documents are present • if the Cryptographic Module isn't a certified signature product (e. g. according to BSI-TR-03112) a suitable test-bed should be used to verify the correctness of the implementation of the signature-related functionality.

Step	Test sequence	Expected Results	Observations
1.	Transfer the archival information package XAIP_OK_Sig_Q / BIN_OK_Sig_Q (see pre-test conditions) to the TOT using the interface function “VerifyRequest”.	The call of the function with this XAIP / BIN as parameter is possible.	
2.	Observe the output of the interface function “VerifyResponse”.	A positive feedback will be received; no error message or error code. A VerificationReport is included in “VerifyResponse”.	
3.	Examine the VerificationReport if the validity verification would be done by the Cryptographic Module.	The validity verification shall be correct and complete, i.e. it includes the entire certificate chain back to a trustworthy root certificate.	
4.	Transfer the archival information package XAIP_OK_Sig_A / BIN_OK_Sig_A (see pre-test conditions) to the TOT. using the interface function “VerifyRequest”.	The call of the function with this XAIP / BIN as parameter is possible.	
5.	Observe the output of the interface function “VerifyResponse”.	A positive feedback will be received; no error message or error code. A VerificationReport is included in “VerifyResponse”.	
6.	Examine the VerificationReport if the validity verification would be done by the Cryptographic Module.	The validity verification shall be correct and complete, i.e. it includes the entire certificate chain back to a trustworthy root certificate.	
7.	Transfer the archival information package XAIP_NOK_Sig_Q / BIN_NOK_SIG (see pre-test conditions) to the TOT using the interface function “VerifyRequest” asking for a verification report.	The call of the function with this XAIP / BIN as parameter is possible.	BIN_NOK_SIG

8.	Observe the output of the interface function “VerifyResponse”.	A negative feedback will be received with error message and error code. A VerificationReport is included in “VerifyResponse”.	
9.	Examine the VerificationReport if the validity verification would be done by the Cryptographic Module.	The Verification Reports includes verification report structures for the signatures, Evidence Records and the XAIP.	
10.	Transfer the archival information package XAIP_NOK_Sig_A / BIN_NOK_SIG (see pre-test conditions) to the TOT. using the interface function “VerifyRequest” asking for a verification report.	The call of the function with this XAIP / BIN as parameter is possible.	
11.	Observe the output of the interface function “VerifyResponse”.	A negative feedback will be received with error message and error code. A VerificationReport is included in “VerifyResponse”.	
12.	Examine the VerificationReport if the validity verification would be done by the Cryptographic Module.	The Verification Reports includes verification report structures for the signatures, Evidence Records and the XAIP.	
13.	Transfer the archival information package XAIP_OK_Sig_Q_ERS (see pre-test conditions) to the TOT using the interface function “VerifyRequest”.	The call of the function with this XAIP as parameter is possible.	
14.	Observe the output of the interface function “VerifyResponse”.	A positive feedback will be received; no error message or error code. A VerificationReport is included in “VerifyResponse”. The verification of the ERs was also successful.	
15.	Examine the VerificationReport if the validity verification would be done by the Cryptographic Module.	The validity verification shall be correct and complete, i.e. it includes the entire certificate chain back to a trustworthy root certificate.	
16.	Transfer the archival information package XAIP_OK_Sig_A_ERS (see pre-test conditions) to the TOT. using the interface function “VerifyRequest”.	The call of the function with this XAIP as parameter is possible.	
17.	Observe the output of the interface function “VerifyResponse”.	A positive feedback will be received; no error message or error code. A VerificationReport is included in “VerifyResponse”. The verification of the ERs was also successful.	
18.	Examine the VerificationReport if the validity verification would be done by the Cryptographic Module.	The validity verification shall be correct and complete, i.e. it includes the entire certificate chain back to a trustworthy root certificate.	
19.	Transfer the archival information package XAIP_NOK_Sig_Q_ERS (see pre-test conditions) to the TOT using the interface function “VerifyRequest” asking for a verification report..	The call of the function with this XAIP as parameter is possible.	

20.	Observe the output of the interface function “VerifyResponse”.	A negative feedback will be received with error message and error code. A VerificationReport is included in “VerifyResponse”.	
21.	Examine the VerificationReport if the validity verification would be done by the Cryptographic Module.	The Verification Reports includes verification report structures for the signatures, Evidence Records and the XAIP.	
22.	Transfer the archival information package XAIP_NOK_Sig_A_ERS (see pre-test conditions) to the TOT using the interface function “VerifyRequest” asking for a verification report.	The call of the function with this XAIP as parameter is possible.	
23.	Observe the output of the interface function “VerifyResponse”.	A negative feedback will be received with error message and error code. A VerificationReport is included in “VerifyResponse”.	
24.	Examine the VerificationReport if the validity verification would be done by the Cryptographic Module.	The Verification Reports includes verification report structures for the signatures, Evidence Records and the XAIP.	
25.	Transfer the archival information package XAIP_NOK_ERS (see pre-test conditions) to the TOT using the interface function “VerifyRequest” asking for a verification report..	The call of the function with this XAIP as parameter is possible.	
26.	Observe the output of the interface function “VerifyResponse”.	A negative feedback will be received with error message and error code. A VerificationReport is included in “VerifyResponse”.	
27.	Examine the VerificationReport if the validity verification would be done by the Cryptographic Module.	The Verification Reports includes verification report structures for the signatures, Evidence Records and the XAIP.	
28.	Transfer the archival information package DXAIP_OK_SIG (see pre-test conditions) to the TOT. using the interface function “VerifyRequest”.	The call of the function with this DXAIP_OK_SIG as parameter is possible.	
29.	Observe the output of the interface function “VerifyResponse”.	A positive feedback will be received; no error message or error code. A VerificationReport is included in “VerifyResponse”. The verification of the DXAIP_OK_SIG was also successful.	
30.	Examine the VerificationReport if the validity verification would be done by the Cryptographic Module.	The validity verification shall be correct and complete, i.e. it includes the entire certificate chain back to a trustworthy root certificate.	
31.	Transfer the archival information package DXAIP_NOK_SIG (see pre-test conditions) to the TOT using the interface function “VerifyRequest” asking for a verification report..	The call of the function with this DXAIP_NOK_OK as parameter is possible.	

32.	Observe the output of the interface function “VerifyResponse”.	A negative feedback will be received with error message and error code. A VerificationReport is included in “VerifyResponse”.	
33.	Examine the VerificationReport if the validity verification would be done by the Cryptographic Module.	The Verification Reports includes verification report structures for the signatures, Evidence Records and the XAIP.	
Verdict			

4.5.1.1.2 S.1.1-02 Verify Request - Unavailable CRL results in invalid certificate

Identifier	S.1.1-02
Requirement	M2:A5.1-16
Test Purpose	If CRLs are used for certificate validation and the CRL is not available or CRL inquiries failed (or the repository, which hosts the CRL cannot accept inquiries), then the respective certificate will be classified as invalid.
Configuration	CONFIG_Common
Pre-test conditions	• Certificate of a Certification Service Provider which support CRL is present

Step	Test sequence	Expected Results	Observations
1.	Sign the XAIP_OK / DXAIP_OK_SIG / BIN archival information package using a valid and not expired certificate issued by a Certification Service Provider which offers CRL.	The signed XAIP_OK / DXAIP_OK_SIG / BIN was created successfully.	
2.	Configure the Cryptographic Module for using CRL.	Configuration of Cryptographic Module was successful.	
3.	Block the network connection to the repository, which hosts the CRL.	The network connection to CRL is blocked.	
4.	Transfer the signed XAIP_OK / DXAIP_OK_SIG / BIN to the TOT using the interface function „VerifyRequest“.	The call of the function with this XAIP_OK / DXAIP_OK_SIG / BIN as parameter is possible.	
5.	Observe the output of the interface function “VerifyResponse”.	A negative feedback will be received; an error message or error code. The certificate was classified as invalid.	

Verdict

4.5.1.2 Sign Request

The test cases M.2-07 (sec. 4.3.7 M.2-07 – Support of Hash functions), M.2-08 (sec. 4.3.8 M.2-08 – Crypto Module uses recommended algorithms for generating signatures), M.2-09 (sec. 4.3.9 M.2-09 – Crypto Module supports canonicalisation for the verification of XML signatures), M.2-10 (sec. 4.3.10 M.2-10 – Canonicalisation procedures do not change the content data), M.2-11 (sec. 4.3.11 M.2-11 – XML-Signatures follow the recommendations of RFC3275), and M.2-12 (sec. 4.3.12 M.2-12 – Reliable verification of electronic signatures) are also relevant here.

4.5.2 Interface S.2

The main purpose of the TR-ESOR-S.2 interface between the ArchiSig-Module and the ECM/long-term storage is to make the necessary read and write access to ArchiSig's own database and the archive database in the ECM/long-term storage possible for the ArchiSig-Module.

This is an interface of a component which is not part of the TR-ESOR middle-ware. Therefore, no conformity tests will be specified here.

4.5.3 Interface S.3

The primary purpose of the TR-ESOR-S.3 interface between the ArchiSig-Module and the Cryptographic-Module is the generation of hash values and the generation and verification of qualified time stamps. Both kinds of data are needed for the development of the Merkle hash trees [MER 1980].

Pre-supposition:

A product which claims to functionally comply with the Interface S.3 specification of this TR has to pass

- all test cases in this section or prove that it supports functional analogous interfaces.

4.5.3.1 Timestamp Request

The test cases M.2-22 (sec. 4.3.22 M.2-22 – Crypto-Module is able to request qualified time stamps), M.2-23 (sec. 4.3.23 M.2-23 – Crypto-Module supports RFC 3161 and suitable algorithms), M.2-24 (sec. 4.3.24 M.2-24 – Time stamps need to bear qualified electronic signature) and M.2-25 (sec. 4.3.25 M.2-25 – Crypto-Module shall verify signatures of received time-stamps) are also relevant here.

4.5.3.2 Verify Request

The test cases of the "VerifyRequest" - function of the interface S.1 (sec. 4.5.1.1 VerifyRequest) are also relevant here.

4.5.3.3 Hash Request

The test cases M.2-07 (sec. 4.3.7 M.2-07 – Support of Hash functions), M.2-08 (sec. 4.3.8 M.2-08 – Crypto Module uses recommended algorithms for generating signatures), M.2-09 (sec. 4.3.9 M.2-09 – Crypto Module supports canonicalisation for the verification of XML signatures), and M.2-10 (sec. 4.3.10 M.2-10 – Canonicalisation procedures do not change the content data) are also relevant here.

4.5.4 Interface S4

The TR-ESOR-S.4 interface should make it possible for the business applications to access the ECM/long-term storage in a standardised and functional manner. Furthermore, the interface should reliably prevent unauthorised access to the ECM/long-term storage.

Note: The term "ArchiSafe" in the following means the logical entry in the archive middleware aside from the actual implementation.

Pre-supposition:

A product which claims to functionally comply with the Interface S.4 specification of this TR has to pass all test cases in this section or to prove that it supports functional analogous interfaces.

4.5.4.1 Archive Submission Request

4.5.4.1.1 S.4.1-01 – Archive Submission Request supports storage of XML-based Archival Information Packages

Identifier	S.4.1-01
Requirement	AF:A3-1 AF:A5-6-1
Test Purpose	The test shall verify that the "ArchiveSubmissionRequest" works well with XAIP format or modified XML formats with the same functionality.
Configuration	CONFIG_ArchiSafe
Pre-test conditions	• The middleware's user manual is available. • If required, perform identification and authentication.

Step	Test sequence	Expected Results	Observations
1.	Compare the description of the XML data format in the middleware's user manual with the XAIP structure described in TR-ESOR Annex TR-ESOR-F.	The implemented XML format complies with the structure defined in TR-ESOR Annex TR-ESOR-F. Deviations are explained and equal functionality is provided. If required, it is explained how a transformation of XAIP to the present XML-format is possible.	
2.	Check the interface functions and their possible parameters.	Data and metadata to be archived shall always be contained in an XML-container and only be passed in this container to the ArchiSafe.	
3.	Store an XAIP_OK_SIG (transformed in the respective XML format) using the "ArchiveSubmissionRequest" function.	The function call is possible.	

4.	Check the output of the “ArchiveSubmissionResponse” function.	The XAIP object is assigned an AOID and stored successfully.	
5.	If the “ArchiveUpdateRequest” function is implemented, use the “ArchiveUpdateRequest” function with the AOID from step 3 to change the data contained within the XAIP.	The function call is possible.	
6.	If the “ArchiveUpdateRequest” function is implemented, check the output of the “ArchiveUpdateResponse” function.	A new version ID is received. The AOID kept identical.	
7.	Use the “ArchiveRetrievalRequest” function with the AOID from step 3 to retrieve the XAIP from the storage.	The function call is possible.	
8.	Check the output of the “ArchiveRetrievalResponse” function.	The archive data object is received in the specified XML format.	
9.	Use the “ArchiveEvidenceRequest” function with the AOID from step 3 to check the XAIPs authenticity and integrity.	The function call is possible.	
10.	Check the output of the “ArchiveEvidenceResponse” function.	An Evidence Record is received.	
11.	If the “ArchiveDataRequest” function is impelmented, use the “ArchiveDataRequest” function with the AOID from step 3 and the dataLocation parameter to identify an individual data element within the XAIP.	The function call is possible.	
12.	If the “ArchiveDataRequest” function is impelmented, check the output of the “ArchiveDataResponse” function.	The requested data value and the corresponding locationValue are received.	
13.	Use the “ArchiveDeletionRequest” function with the AOID from step 3 to delete the XAIP.	The function call is possible.	
14.	Check the result of the “ArchiveDeletionResponse” function by attempting to retrieve the deleted XAIP calling the “ArchiveRetrievalRequest” with the corresponding AOID as parameter.	The XAIP has been deleted from the storage.	
Verdict			

4.5.4.1.2 S.4.1-02 – Archive Submission yields unique AOID

Identifier	S.4.1-02
Requirement	MD:A5.1-4 M1:A4.1-6 M1:A4.1-7 M3:A4.2-2
Test Purpose	The test shall verify that a unique, unchangeable AOID is assigned to each archive data object that is stored in the ECM. The test shall verify that an already archived object will not be overwritten or changed by an “ArchiveSubmissionRequest”.
Configuration	CONFIG_ArchiSafe
Pre-test conditions	• Tester has write permissions on the system • If required, perform identification and authentication

Step	Test sequence	Expected Results	Observations
1.	Transfer an XAIP_OK or BIN to the TOT using the interface function “ArchiveSubmissionRequest”.	The call of the function with this XAIP / BIN as a parameter is possible.	
2.	Observe the output of the interface function “ArchiveSubmissionResponse”.	A positive feedback is received. No error message or error code occurs. An AOID is assigned to the XAIP / BIN.	
3.	Transfer the archival information package XAIP_OK / BIN to the TOT using the interface function “ArchiveSubmissionRequest”.	The call of the function with this XAIP / BIN as a parameter is possible.	
4.	Observe the output of the interface function “ArchiveSubmissionResponse”.	A positive feedback is received. No error message or error code occurs. An AOID is assigned to the XAIP.	
5.	Compare the AOIDs.	The AOIDs are not equal.	
6.	Transfer the very same XAIP_OK or BIN from step 1 to the TOT using the interface function “ArchiveSubmissionRequest”.	The call of the function with this XAIP as a parameter is possible.	
7.	Observe the output of the interface function “ArchiveSubmissionResponse”.	A positive feedback is received. No error message or error code occurs. Another AOID is assigned to the XAIP / BIN than in step 2.	
8.	Retrieve the XAIP_OK's with the AOID's from step 2 and 4.	Both XAIP's could be retrieved. They are identical except the AOID (and maybe some other metadata like date and time of archival).	

9.	If the TOT supports “ArchiveUpdateRequest”, update one XAIP_OK or XAIP(BIN) by using the “ArchiveUpdateRequest” and the AOID from step 2.	The update is successful.	
10.	Retrieve the XAIP_OK's with the AOID's from step 2 and 7.	Both XAIPs could be retrieved. They are not identical. The second XAIP includes the update whereas the first XAIP is still unchanged.	
11.	Transfer an XAIP_OK or BIN to the TOT using the interface function “ArchiveSubmissionRequest” together with another collision free AOID, created by the client application, which was not used before.	The call of the function with this XAIP / BIN as a parameter is possible.	
12.	Observe the output of the interface function “ArchiveSubmissionResponse”.	A positive feedback is received. No error message or error code occurs. The AOID from step 11 is assigned to the XAIP / BIN.	
13.	Use the “ArchiveRetrievalRequest” function with the AOID from step 11 to retrieve the XAIP_OK or XAIP(BIN) from the storage.	The function call is possible without an error message. The stored XAIP_OK or XAIP(BIN) will be returned in a XAIP format.	
Verdict			

4.5.4.1.3 S.4.1-03 – Archive Submission with valid binary object is possible

Identifier	S.4.1-03
Requirement	MD:A5.1-4 M1:A4.1-1 M3:A4.2-2
Test Purpose	The test shall verify that a binary document can be stored in the ECM/long-term storage and the call returns a unique AOID Note: If the interface S.4 supports "ArchiveSubmissionRequests" for XAIPs only, the test will be considered as successfully passed.
Configuration	CONFIG_ArchiSafe
Pre-test conditions	• If required, establish a session with the TOT in order to perform the following tests • If required, perform identification and authentication

Step	**Test sequence**	**Expected Results**	**Observations**
1.	Transfer several documents BIN to the TOT using the interface function "ArchiveSubmissionRequest".	The call of the function with this document as a parameter is possible.	
2.	Observe the output of the interface function "ArchiveSubmissionResponse".	A positive feedback is received. No error message or error code occurs. An unique AOID is assigned to each and every object.	
3.	Check the log files of the TOT for a record about an XML schema check.	There is no record about an XML schema verification of this document.	
4.	Use the "ArchiveRetrievalRequest" function with the AOID from step 2 to retrieve the binary object from the storage.	The function call is possible without an error message. The stored binary object will be returned as an XAIP(BIN).	
Verdict			

4.5.4.1.4 S.4.1-04 – Archive Submission is always possible via a secure communication channel

Identifier	S.4.1-04
Requirement	MD:A5.1-2
Test Purpose	The test shall verify whether the storage of electronic documents and data from external IT applications is always possible via a securecommunication channel.
Configuration	CONFIG_ArchiSafe
Pre-test conditions	• The IT system documentation is available • If required, perform identification and authentication • Administration access to the IT systems is needed

Step	Test sequence	Expected Results	Observations
1.	Check whether a secure communication channel between upstream application and TOT is configured and activated.	A secure communication channel is set up and active.	
2.	Start logging the data traffic between the external IT application and the middleware.	The data logging process has been started.	
3.	Store an XAIP_OK_SIG or BIN from the external IT application via the middleware to the ECM.	The function call is possible.	
4.	Close the connection of the two components. Stop logging the data traffic.	The complete data exchange between the components has been intercepted and logged.	
5.	Check the data traffic log file for unprotected document data.	No document data can be accessed.	
Verdict			

4.5.4.1.5 S.4.1-05 – Archive Submission includes signature verification and storage of results

Identifier	S.4.1-05
Requirement	M1:A4.1-2 M1:A4.1-3 M3:A4.1-1
Test Purpose	The test shall verify that the ArchiSafe module is able to initiate the verification of electronic signatures of the XAIPs or BINs before they are stored and that an error message is received in the case of a failed signature check. The test shall verify that it is possible for the ArchiSafe module to enter signature verification results including the associated certificate information into the archive object.
Configuration	CONFIG_ArchiSafe
Pre-test conditions	• Tester has Read/Write permissions on the system • Perform authentication if necessary

Step	**Test sequence**	**Expected Results**	**Observations**
1.	Verify that the configuration of the ArchiSafe module enables the automatic signature check while submitting an archive object.	Automatic signature check can be enabled and is enabled.	
2.	Store an XAIP_OK_SIG or BIN to the TOT using the interface function “ArchiveSubmissionRequest”.	The call of the function is possible.	
3.	Observe the output of the interface function “ArchiveSubmissionResponse”.	A positive feedback is received. No error message or error code occurs. An AOID is assigned to the stored archive object.	
4.	Store an XAIP_NOK_SIG or BIN_NOK_SIG to the TOT using the interface function “ArchiveSubmissionRequest”.	The call of the function is possible.	
5.	Observe the output of the interface function “ArchiveSubmissionResponse”.	A negative feedback will be received. An error message or error code occurs. The log file contains an error message with a signature. The archive object may be stored and an AOID may be returned.	
6.	Retrieve the XAIP_OK_SIG by using the “ArchiveRetrievalRequest” function and the AOID from step 3.	The XAIP_OK_SIG is retrieved.	
7.	Check the XAIP_OK_SIG, especially the credential section, whether the signature verification information	The certificates, certification verification information and the signature verification information are included in the retrieved	

	are included.	XAIP_OK_SIG.	
8.	If archived/stored, retrieve the XAIP_NOK_SIG by using the “ArchiveRetrievalRequest” function and the AOID from step 5.	The XAIP_NOK_SIG is retrieved.	
9.	Check the XAIP_NOK_SIG, especially the credential section, whether the signature verification information are included.	The certificates, certification verification information and the signature verification information are included in the retrieved XAIP_NOK_SIG.	
10.	Retrieve the XAIP(BIN) by using the “ArchiveRetrievalRequest” function and the AOID from step 7.	The XAIP(BIN) is retrieved in the XAIP format including all assigned metadata and the BIN data as content.	
11.	Check the retrieved XAIP and all the metadata whether the signature verification information are included.	The certificates, certification verification information and the signature verification information are included in the retrieved XAIP	
12.	If archived/stored, retrieve the BIN_NOK_SIG by using the “ArchiveRetrievalRequest” function and the AOID from step 9.	The BIN_NOK_SIG is retrieved in the XAIP format including all assigned metadata and the BIN data as content.	
13.	Check the retrieved XAIP and all the metadata whether the signature verification information are included.	The certificates, certification verification information and the signature verification information are included in the retrieved XAIP	
14.	Store an XAIP_OK_SIG_OK_ER to the TOT using the interface function “ArchiveSubmissionRequest”.	The call of the function is possible.	
15.	Observe the output of the interface function “ArchiveSubmissionResponse”.	A positive feedback is received. No error message or error code occurs. An AOID is assigned to the stored archive object.	
16.	Retrieve the XAIP_OK_SIG_OK_ER by using the “ArchiveRetrievalRequest” function and the AOID from step 15.	The XAIP_OK_SIG_OK_ER is retrieved.	
17.	Check the XAIP_OK_SIG_OK_ER, especially the credential section, whether the signature verification information and evidence record verification information are included.	The certificates, certification verification information and the signature verification information and evidence record verification information are included in the retrieved XAIP_OK_SIG_OK_ER.	
18.	Store an XAIP_NOK_SIG_OK_ER to the TOT using the interface function “ArchiveSubmissionRequest”.	The call of the function is possible.	
19.	Observe the output of the interface function “ArchiveSubmissionResponse”.	A negative feedback will be received. An error message or error code occurs. The log file contains an error message with a signature.	

		The archive object may be stored and an AOID may be returned.	
20.	If archived/stored, retrieve the XAIP_NOK_SIG_OK_ER by using the “ArchiveRetrievalRequest” function and the AOID from step 19.	The XAIP_NOK_SIG_OK_ER is retrieved in the XAIP format.	
21.	Check the retrieved XAIP and all the metadata whether the signature verification information and the evidence record verification information are included.	The certificates, certification verification information and the signature verification information and the evidence record verification information are included in the retrieved XAIP	
22.	Store an XAIP_NOK_ER to the TOT using the interface function “ArchiveSubmissionRequest”.	The call of the function is possible.	
23.	Observe the output of the interface function “ArchiveSubmissionResponse”.	A negative feedback will be received. An error message or error code occurs. The log file contains an error message with a signature. The archive object may be stored and an AOID may be returned.	
24.	If archived/stored, retrieve the XAIP_NOK_ER by using the “ArchiveRetrievalRequest” function and the AOID from step 23.	The XAIP_NOK_ER is retrieved in the XAIP format.	
25.	Check the retrieved XAIP and all the metadata whether the signature verification information and the evidence record verification information are included.	The certificates, certification verification information and the signature verification information and the evidence record verification information are included in the retrieved XAIP	
Verdict			

4.5.4.1.6 S.4.1-06 – Archive Submission Request does not change the data objects within the XAIP or BIN

Identifier	S.4.1-06
Requirement	M1:A4.1-5
Test Purpose	The test shall verify that the ArchiSafe module does not change the primary data objects within the XAIPs or BINs.
Configuration	CONFIG_ArchiSafe
Pre-test conditions	• If required, perform identification and authentication

Step	Test sequence	Expected Results	Observations
1.	Store an XAIP_OK_SIG or BIN using the interface function “ArchiveSubmissionRequest”.	The call of the function is possible.	
2.	Observe the output of the interface function “ArchiveSubmissionResponse”.	A positive feedback is received. No error message or error code occurs. An AOID is assigned to the XAIP / BIN.	
3.	Request the XAIP with the “ArchiveRetrievalRequest” function and the AOID from step 2.	The call of the function is possible.	
4.	Compare the data objects of the retrieved XAIP with the data objects of the XAIP / BIN that has originally been stored in step 1.	The data objects are identical.	
5.	Check vendor documentation whether ArchiSafe resp. the TOT provides any function to modify the actual primary data content or whether a conversion of the primary data content is required.	No such function or requirement exists.	

Verdict

4.5.4.1.7 S.4.1-07 – Archive Submission of invalid XML data is not possible

Identifier	S.4.1-07
Requirement	MD:A5.1-4 M1:A4.1-1
Test Purpose	The test shall verify that it is not possible to store an archival information package with a wrong XML syntax.
Configuration	CONFIG_ArchiSafe (includes XSD schema verification enabled).
Pre-test conditions	• If required, establish a session with the TOT in order to perform the following tests • If required, perform identification and authentication

Step	**Test sequence**	**Expected Results**	**Observations**
1.	Transfer the archival information package XAIP_NOK to the TOT using the interface function “ArchiveSubmissionRequest”.	The call of the function with this XAIP as a parameter is possible.	
2.	Observe the output of the interface function “ArchiveSubmissionResponse”.	A clear and understandable error message or error code will be received.	
3.	Check the log files of the TOT for an error record about the XML schema check.	There is an error record showing that the XML schema verification of this XAIP failed.	
4.	Check whether the XAIP is stored.	The XAIP is not stored.	
Verdict			

4.5.4.1.8 S.4.1-08 – Application protocol uses request-response-message-exchange pattern

Identifier	S.4.1-08
Requirement	AF:A5.6-7
Test Purpose	The test shall verify that a protocol within the secure Communication Channel is used by which, among other things, the technical confirmation of the receipt of a client request is realised.
Configuration	CONFIG_ArchiSafe
Pre-test conditions	• The IT system documentation is available • If required, perform identification and authentication • Administration access to the IT systems is needed

Step	**Test sequence**	**Expected Results**	**Observations**
1.	Check the IT system documentation for the used protocol within the secure communication channel protocol.	The documentation states which protocol is used (e.g. HTTP, RPC, RMI, . . .).	
2.	Check the documentation for this protocol whether technical confirmations of receipts are implemented.	The protocol implements such confirmations (e.g. TCP ACK, HTTP Return codes, . . .).	
Verdict			

4.5.4.1.9 S.4.1-10 – WSDL and Document literal encoding for SOAP should be used

Identifier	S.4.1-10
Requirement	AF:A5.6-8
Test Purpose	The test shall verify whether SOAP Document Literal Encoding is used and if the external interfaces of all archive system components are published via WSDL.
Configuration	CONFIG_ArchiSafe
Pre-test conditions	• The middleware documentation is available. • The application documentation is available.

Step	**Test sequence**	**Expected Results**	**Observations**
1.	Check the middleware documentation for the use of WSDL.	WSDL is used to publish the external interfaces of all archive system components.	
2.	Check the middleware documentation for the use of SOAP Document Literal Encoding.	SOAP Document Literal Encoding is used.	
Verdict			

4.5.4.2 Archive Update Request

Pre-supposition:

A product which claims to to comply with the update functionality according to M.1-04 and S.4.2-01 “ArchiveUpdateRequest” of this TR has to pass the following test case or prove that it supports functional analogous functions.

4.5.4.2.1 S.4.2-01 – Archive Update Request is possible and ArchiSig immediately secures the new object

Identifier	S.4.2-01
Requirement	MD:A5.1-11 M1:A4.2-1 M1:A4.2-8
Test Purpose	The test shall verify that an XAIP with a correct XML structure or a BIN archive object are correctly stored in the ECM/long-term storage. The test shall check that an XAIP / BIN will be send to the ArchiSig module before it will be stored in the ECM/long-term storage. (Archive Submission & Archive Update). The test shall check, if for each XAIP / BIN stored in the ECM/long-term storage a unique AOID will be generated and returned.
Configuration	CONFIG_ArchiSafe
Pre-test conditions	• If required, establish a session with the TOT in order to perform the following tests • If required, perform identification and authentication

Step	**Test sequence**	**Expected Results**	**Observations**
1.	Transfer several XAIP_OK / BIN to the TOT using the interface function “ArchiveSubmissionRequest”.	The call of the function with this XAIP / BIN as a parameter is possible.	
2.	Observe the output of the interface function “ArchiveSubmissionResponse”.	A positive feedback is received. No error message or error code occurs. A unique AOID is assigned to each and every XAIP / BIN.	
3.	Check the log files of the TOT for a record about the XML schema check. In the case of storing BINs skip this step.	There is a record showing the positive XML schema verification of the XAIP.	
4.	Use a number of “ArchiveRetrievalRequest” functions with the AOIDs from step 2 as parameters.	The call of the functions with this AOIDs as parameters is possible.	
5.	Observe the output of the interface functions “ArchiveRetrievalResponse”.	A positive feedback is received. No error message or error code occurs. The originally stored XAIPs or XAIPs which embody the BINs were retrieved (XAIP(BIN)s).	
6.	Compare the retrieved XAIPs and the XAIPs, resp. the embodied BINs and the BINs stored in step 1.	The contents are identical. The retrieved XAIPs contain additionally the respective AOID.	

		The original XAIPs do not contain this AOID.	
7.	*Between execution of step 1 and step 7 must be less time as ArchiSig is configured to perform automated signature renewal because it should be checked whether newly submitted archive objects run through the ArchiSig module and initial archive time stamps will be generated immediately.* Using several calls, request the ERS records for the XAIPs / BINs stored in step 1 using the AOIDs from step 2 as a parameter.	The ERS records can be received, even if the archive object was submitted just very shortly before this test step.	
8.	Check whether the hash values in the ERs for the XAIPs / BINs refer to the XAIPs/BINs with the AOID included. In case of doubt, recalculate the hash values for the XAIPs / BINs with the AOID (see M.3 sec. 2.4.1 for details) and compare that with the hash values listed in the ERS records.	The hash values listed in the ERS records refer to the XAIPs / BINs with the AOIDs included. The hash values for this XAIPs/BINs are correctly mentioned in the ERS records.	
9.	Repeat the steps 1-8 immediately in order to be sure that ArchiSig did not perform an Archive Time Stamp renewal between step 1 and 7.	Same results as expected above.	
10.	Repeat the steps 1-9 but instead of submit use the “ArchiveUpdateRequest” function.	Update is successful, a version ID will be issued and returned. The log records show the XML schema check for storing each XAIP/XAIP(BIN) The updated XAIPs will be retrieved. The retrieved XAIPs contain the requested changes/updates. The ERSs can be retrieved. The hash values identifie the updated XAIPs / XAIP(BIN)s. Same results in the repetition.	
Verdict			

4.5.4.2.2 S.4.2-02 – Archive Update requires existing AOID

Identifier	S.4.2-02
Requirement	M1:A4.2-1
Test Purpose	The test shall verify that the ArchiSafe module can only update an archive data object when a valid and existing AOID is part of the update request.
Configuration	CONFIG_ArchiSafe
Pre-test conditions	• If required, perform identification and authentication

Step	**Test sequence**	**Expected Results**	**Observations**
1.	Try to issue an “ArchiveUpdateRequest” with an AOID that does not exist.	The function call is possible.	
2.	Observe the output of the interface function “ArchiveUpdateResponse”.	An error message or error code is received.	
Verdict			

4.5.4.2.3 S.4.2-03 – Archive Update is allowed and results in a new version ID

Identifier	S.4.2-03
Requirement	MD:A5.1-11 MD:A5.1-14 M1:A4.2-5 M1:A4.2-9
Test Purpose	The test shall verify whether it is possible to change documents and data including the associated meta data. If archive objects are updated, a new version ID is to be issued.
Configuration	CONFIG_ArchiSafe
Pre-test conditions	• Tester has read / write permissions on the Middleware • If required, perform identification and authentication

Step	Test sequence	Expected Results	Observations
1.	Check if the interface function “ArchiveUpdateRequest” exists.	The function exists.	
2.	Submit an XAIP_OK or BIN with data to the TOT using the interface function “ArchiveSubmissionRequest”.	The call of the function with this XAIP / BIN as a parameter is possible.	
3.	Observe the output of the interface function “ArchiveSubmissionResponse”.	A positive feedback is received. No error message or error code occurs. An AOID is assigned.	
4.	Using the interface function “ArchiveUpdateRequest” and the AOID from step 3 to **add** additional content to the XAIP / XAIP(BIN).	The call of the function with this binary data and the AOID as parameters is possible.	
5.	Observe the output of the interface function “ArchiveUpdateResponse”.	A positive feedback is received. No error message or error code occurs. A new Version ID is received.	
6.	Using the interface function “ArchiveUpdateRequest” and the AOID from step 3 and the Version ID from step 5 to **add** additional metadata to the XAIP / XAIP(BIN).	The call of the function with this data and the AOID as parameters is possible.	
7.	Observe the output of the interface function “ArchiveUpdateResponse”.	A positive feedback is received. No error message or error code occurs. A new Version ID is received.	
8.	Using the interface function “ArchiveUpdateRequest” and the AOID from step 3 and the Version ID from step 7 to **update** content of the XAIP / XAIP(BIN).	The call of the function with this data and the AOID as parameters is possible.	

9.	Observe the output of the interface function “ArchiveUpdateResponse”.	A positive feedback is received. No error message or error code occurs. A new Version ID is received.	
10.	Using the interface function “ArchiveUpdateRequest” and the AOID from step 3 and the new Version ID from step 9 to **update** metadata of the XAIP / XAIP(BIN).	The call of the function with this data and the AOID as parameters is possible.	
11.	Observe the output of the interface function “ArchiveUpdateResponse”.	A positive feedback is received. No error message or error code occurs. A new Version ID is received.	
12.	Using the interface function “ArchiveUpdateRequest” and the AOID from step 3 and the new Version ID from step 11 to **remove**[11] one piece of data from the XAIP / XAIP(BIN), not the complete XAIP.	The call of the function with this data and the AOID as parameters is possible.	
13.	Using the interface function “ArchiveUpdateRequest” and the AOID from step 3 and the new Version ID from step 7 to **update** metadata of the XAIP / XAIP(BIN).	The call of the function with this data and the AOID as parameters is possible.	
14.	Observe the output of the interface function “ArchiveUpdateResponse”.	A negative feedback will be received. An error message or error code occurs. The log file contains an error message indicating the wrong Version ID. The updated archive object is not stored.	
15.	Observe the output of the interface function “ArchiveUpdateResponse”.	A positive feedback is received. No error message or error code occurs. A new Version ID is received.	
16.	Retrieve the XAIP using the the AOID from step 3 and check whether all changes are reflected.	The retrieved versions of the XAIP reflect all changes made in the XAIP or XAIP(BIN). Especially a version manifest per version exists.	
17.	Check the log file for logs of the changes and update procedures.	The log files contain messages about all the changes.	
Verdict			

4.5.4.3 S.4.2-04 – Archive Update requires data and creates new version

Identifier	S.4.2-04
Requirement	MD:A5.1-14

[11] This „remove“ means that the element is not longer part of the most current version of the XAIP. Nevertheless, the element is still stored in the XAIP for evidence purposes. If an older version of the XAIP would be requested, the element would be included and available.

	M1:A4.2-2 M1:A4.2-7
Test Purpose	The test shall verify that the ArchiSafe module can only update an archive data object when the data object or meta data that should be updated are part of the request and not empty and that the original data object is not changed but a new version of the XAIP is created.
Configuration	CONFIG_ArchiSafe
Pre-test conditions	• If required, perform identification and authentication

Step	Test sequence	Expected Results	Observations
1.	Store an XAIP_OK_SIG or BIN using the interface function “ArchiveSubmissionRequest”.	The call of the function is possible.	
2.	Observe the output of the interface function “ArchiveSubmissionResponse”.	A positive feedback is received. No error message or error code occurs. An AOID is assigned to the archived XAIP / BIN.	
3.	Try to update this XAIP or BIN using the interface function “ArchiveUpdateRequest” with the AOID from step 2 without any data object as a parameter.	The call of the function should be possible.	
4.	Observe the output of the interface function “ArchiveUpdateResponse”.	An error message or error code will be received.	
5.	Try to update the archived data object using the interface function “ArchiveUpdateRequest” with the AOID from step 2 with an empty DXAIP_NOK .	The call of the function should be possible.	
6.	Observe the output of the interface function “ArchiveUpdateResponse”.	An error message or error code will be received.	
7.	Try to update the archived data object using the interface function “ArchiveUpdateRequest” with the AOID from step 2 with a valid DXAIP_OK on base of a valid XAIP_OK/ XAIP(BIN).	The call of the function should be possible.	
8.	Observe the output of the interface function “ArchiveUpdateResponse”.	A positive feedback is received. No error message or error code occurs. A new Version ID is received.	
9.	Retrieve the originally stored version by issuing an “ArchiveRetrievalRequest” with the AOID from step 2 with the very first version ID (e.g. v1).	The call of the function is possible.	
10.	Observe the output of the interface function “ArchiveRetrievalResponse”.	The original, unchanged version of the XAIP / BIN (embedded in an XAIP (XAIP(BIN))) is successfully retrieved.	

11.	Retrieve the originally stored version by issuing an “ArchiveRetrievalRequest” with the AOID from step 2 without a version ID.	The call of the function is possible.	
12.	Observe the output of the interface function “ArchiveRetrievalResponse”.	The most current, changed version of the XAIP/ XAIP(BIN) (embedded in an XAIP) is successfully retrieved.	
13.	Retrieve all stored versions by issuing an “ArchiveRetrievalRequest” with the AOID from step 2 with the version ID “all”.	The call of the function is possible.	
14.	Observe the output of the interface function “ArchiveRetrievalResponse”.	All versions of the XAIP/ XAIP(BIN) (embedded in an XAIP) is successfully retrieved.	
Verdict			

4.5.4.3.1 S.4.2-05 – Only authorised entities can change data

Identifier	S.4.2-05
Requirement	MD:A5.1-12
Test Purpose	The test shall verify that changes to documents and data including the associated meta data is not possible for unauthorised users or applications.
Configuration	CONFIG_ArchiSafe
Pre-test conditions	• Tester has no read/write permissions on the middleware • Do not perform any authentication against ArchiSafe

Step	Test sequence	Expected Results	Observations
1.	Submit a XAIP_OK or BIN to the middleware using an **account A** from a client A (if TOT is multi-client-capable). Perform authentication when required.	The XAIP / BIN was archived. An AOID was returned.	
2.	Retrieve a XAIP using the AOID and an **account A** from a client A (if TOT is multi-client-capable). Perform authentication when required.	XAIP could be retrieved.	
3.	Update the XAIP / XAIP(BIN) several times using the AOID and an **account A** from a client A (if TOT is multi-client-capable). Perform authentication when required.	All updates are successfully performed.	
4.	Disconnect from the TOT.	Any existing secure channels are terminated.	
5.	Reconnect to the TOT and try to retrieve a XAIP using the AOID and an **account B** from a client A (if TOT is multi-client-capable). Perform authentication when required.	Access denied.	
6.	Update the XAIP / XAIP(BIN) using the AOID and an **account B** from a client A (if TOT is multi-client-capable). Perform authentication when required.	Access denied.	
7.	Retrieve a XAIP using the AOID and an **account A** from a **client B** (if TOT is multi-client-capable). Perform authentication when required.	Access denied.	

8.	Update the XAIP / XAIP(BIN) using the AOID and an **account A** from a **client B** (if TOT is multi-client-capable). Perform authentication when required.	Access denied.	
Verdict			

4.5.4.3.2 S.4.2-06 – Signature and data format checks are also performed on update

Identifier	S.4.2-06
Requirement	MD:A5.1-13M1:A4.2-4 M1:A4.2-7
Test Purpose	The test shall verify that the same data format and signature checks that are performed for the archival of documents and XAIPs are also performed when already archived XAIPs are changed.
Configuration	CONFIG_ArchiSafe
Pre-test conditions	• Tester has write permissions on the Middleware • If required, perform identification and authentication

Step	Test sequence	Expected Results	Observations
1.	Perform test case S.4.1-07 but with "ArchiveUpdateRequest" instead of "ArchiveSubmissionRequest".	For updates also the XML schema validation will be performed.	
2.	Perform test case S.4.1-05 but with "ArchiveUpdateRequest" instead of "ArchiveSubmissionRequest". Add a signed data object to an already archived XAIP.	The added signatures of signed data objects will also be validated.	
Verdict			

4.5.4.3.3 S.4.2-07 – All updates shall be traceable and keep the previous version untouched

Identifier	S.4.2-07
Requirement	MD:A5.1-14 M1:A4.2-7
Test Purpose	The test shall verify whether all changes are traceable and that changes to archived XAIPs/BINs are only applied to the new versions while leaving the existing versions untouched.
Configuration	CONFIG_ArchiSafe
Pre-test conditions	• The tester has read/write permissions on the middleware • If required, perform identification and authentication

Step	Test sequence	Expected Results	Observations
1.	Store an XAIP_OK_SIG / BIN with data to the TOT using the interface function “ArchiveSubmissionRequest”.	The call of the function with this XAIP / BIN as a parameter is possible.	
2.	Observe the output of the interface function “ArchiveSubmissionResponse”.	A positive feedback is received. No error message or error code occurs. An AOID is assigned.	
3.	Using the interface function “ArchiveUpdateRequest“ and the AOID from step 2, add a few changes to the XAIP_OK_SIG / XAIP(BIN) .	The call of the function with this XAIP / XAIP(BIN) and the AOID as parameters is possible.	
4.	Observe the output of the interface function “ArchiveUpdateResponse”.	A positive feedback is received. No error message or error code occurs. A new Version ID is received.	
5.	Request a XAIP from the TOT using the interface function “ArchiveRetrievalRequest“ with the AOID from step 2 and the version ID as parameters which indicates the very first version.	The call of the function with this AOID and the Version ID as parameters is possible.	
6.	Observe the output of the interface function “ArchiveRetrievalResponse”.	A positive feedback is received. No error message or error code occurs. An XAIP is received.	
7.	Compare the retrieved XAIP with the XAIP stored in step 1.	The XAIP, resp. the BIN embedded in the retrieved XAIP, is the same file that was stored in step 1.	
8.	Request the XAIP from the TOT using the interface function “ArchiveRetrievalRequest“ with the AOID from step 2 and a valid version ID which is not the very first and not the very last version ID.	The call of the function with this AOID as a parameter is possible.	

9.	Observe the output of the interface function “ArchiveRetrievalResponse”.	A positive feedback is received. No error message or error code occurs. An XAIP is received.	
10.	Compare the retrieved XAIP with the XAIP stored in step 1 and all the changes done in step 3.	The XAIP reflects all changes done in step 3 as appropriate for the selected version ID. Especially, XAIP does not contain the changes which are applied to newer versions that the version selected.	
11.	Request the XAIP from the TOT using the interface function “ArchiveRetrievalRequest“ with the AOID from step 2 and without a version ID.	The call of the function with this AOID as a parameter is possible.	
12.	Observe the output of the interface function “ArchiveRetrievalResponse”.	A positive feedback is received. No error message or error code occurs. An XAIP is received.	
13.	Compare the retrieved XAIP with the XAIP stored in step 1 and all the changes done in step 3.	The XAIP reflects all changes done in step 3.	
Verdict			

4.5.4.3.4 S.4.2-08 – Update shall not impair the probative value

Identifier	S.4.2-08
Requirement	MD:A5.1-15 M1:A4.2-7
Test Purpose	The test shall verify that the probative value is not compromised by changes.
Configuration	CONFIG_ArchiSafe
Pre-test conditions	• Tester has read/write and administrative permissions on the Middleware • Test S.4.2-07 has been performed successfully • If required, perform identification and authentication

Step	Test sequence	Expected Results	Observations
1.	Store an XAIP_OK or BIN to the TOT using the interface function “ArchiveSubmissionRequest”.	The call of the function with this XAIP / BIN as a parameter is possible.	
2.	Observe the output of the interface function “ArchiveSubmissionResponse”.	A positive feedback is received. No error message or error code occurs. An AOID is assigned.	
3.	Request Evidence Records using the AOID from step 2 and the interface function „ArchiveEvidenceRequest“.	The call of the function with this AOID as a parameter is possible.	
4.	Observe the output of the interface function “ArchiveEvidenceResponse”.	A positive feedback is received. No error message or error code occurs. An Evidence Record is received for the first version ID (e.g. “v1”).	
5.	Verify the retrieved ERs by using an appropriate tool.	The tool shows that the ERs is upright.	
6.	Change the hash algorithm.	The hash algorithm is changed.	
7.	Initiate the hash-tree renewal process.	The re-hash process is initiated.	
8.	Using the interface function „ArchiveUpdateRequest“ and the AOID from step 2 add additional changes to XAIP_OK / XAIP(BIN).	The call of the function with this AOID and binary data as parameters is possible.	
9.	Observe the output of the interface function “ArchiveUpdateResponse”.	A positive feedback is received. No error message or error code occurs. A new augmented Version ID is assigned (e.g. “v2”).	

10.	Request the XAIP using the AOID from step 2 and the interface function „ArchiveRetrievalRequest“.	The call of the function with this AOID as a parameter is possible.	
11.	Observe the output of the interface function “ArchiveRetrievalResponse”.	A positive feedback is received. No error message or error code occurs. An XAIP is received.	
12.	Request Evidence Records using the AOID from step 2 and the interface function „ArchiveEvidenceRequest“.	The call of the function with this AOID as a parameter is possible.	
13.	Observe the output of the interface function “ArchiveEvidenceResponse”.	A positive feedback is received. No error message or error code occurs. An Evidence Record is received for the new augmented version ID (e.g. “v2”).	
14.	Verify the retrieved ERs by using an appropriate tool.	The tool shows that the ERs are upright.	
15.	Using the interface function “ArchiveUpdateRequest“ and the AOID from step 2 to change the XAIP_OK /XAIP(BIN) (add metadata) using DXAIP_OK.	The call of the function with this AOID and the DXAIP_OK as parameters is possible.	
16.	Observe the output of the interface function “ArchiveUpdateResponse”.	A positive feedback is received. No error message or error code occurs. A new Version ID is assigned.	
17.	Request the XAIP with the AOID from step 2 and the interface function „ArchiveRetrievalRequest“.	The call of the function with this AOID as a parameter is possible.	
18.	Observe the output of the interface function “ArchiveRetrievalResponse”.	A positive feedback is received. No error message or error code occurs. An XAIP is received.	
19.	Request Evidence Records using the AOID from step 2 and the interface function „ArchiveEvidenceRequest“.	The call of the function with this AOID as a parameter is possible.	
20.	Observe the output of the interface function “ArchiveEvidenceResponse”.	A positive feedback is received. No error message or error code occurs. An Evidence Record is received for the new augmented version ID (e.g. “v3”).	
21.	Verify the retrieved ERs by using an appropriate tool.	The tool shows that the ERs are integer.	
22.	Compare the ERs from step 17 with the ERs from step 10.	The evidence data from step 17 differs from the evidence data retrieved in step 10.	
23.	Using the interface function “ArchiveUpdateRequest“ and the AOID from step 2 to delete the changes to XAIP_OK /XAIP(BIN) added in step 5.	The call of the function with this AOID and binary data as parameters is possible.	
24.	Observe the output of the interface function “ArchiveUpdateResponse”.	A positive feedback is received. No error message or error code occurs. A new Version ID is assigned.	

25.	Request the XAIP with the AOID from step 2 and the interface function „ArchiveRetrievalRequest“.	The call of the function with this AOID as a parameter is possible.	
26.	Observe the output of the interface function “ArchiveRetrievalResponse”.	A positive feedback is received. No error message or error code occurs. An XAIP is received.	
27.	Request Evidence Records using the AOID from step 2 and the interface function „ArchiveEvidenceRequest“.	The call of the function with this AOID as a parameter is possible.	
28.	Observe the output of the interface function “ArchiveEvidenceResponse”.	A positive feedback is received. No error message or error code occurs. An Evidence Record is received for new augmented version ID (e.g. “v4”).	
29.	Calculate manually the evidence data for the updated XAIP / XAIP(BIN). For this purpose use the time stamp information provided in the ERs retrieved in the previous step.	The evidence data has been calculated.	
30.	Compare the manually calculated evidence data with the evidence data of the requested Evidence Record.	The evidence data is equal but differs from the evidence data retrieved in step 18.	
31.	Request Evidence Records using the AOID from step 2 and the interface function „ArchiveEvidenceRequest“ for all Version Ids (e.g. Version ID = “all”).	The call of the function with this AOID as a parameter is possible.	
32.	Observe the output of the interface function “ArchiveEvidenceResponse”.	A positive feedback is received. No error message or error code occurs. The Evidence Records for the previously created four versions are received.	
Verdict			

4.5.4.3.5 S.4.2-09 – Update can not delete data / Versions can be retrieved separately

Identifier	S.4.2-09
Requirement	MD:A5.1-16 MD:A5.1-20 M1:A4.2-2 M1:A4.2-7
Test Purpose	The test shall verify that the update function cannot be used to completely and ultimately delete any data, meta data or complete XAIPs/BINs. The test shall verify that it is possible to retrieve each version of a changed data structure individually by using the version ID as a parameter when issuing the archive retrieval request.
Configuration	CONFIG_ArchiSafe
Pre-test conditions	• Tester has read/write permissions on the Middleware • If required, perform identification and authentication

Step	Test sequence	Expected Results	Observations
1.	Store an XAIP_OK or BIN with data to the TOT using the interface function “ArchiveSubmissionRequest”.	The call of the function with this XAIP / BIN as a parameter is possible.	
2.	Observe the output of the interface function “ArchiveSubmissionResponse”.	A positive feedback is received. No error message or error code occurs. An AOID is assigned and returned.	
3.	Using the interface function “ArchiveUpdateRequest” and the AOID from step 2 add an additional data element to the already existing archive data object	The call of the function with a data element and the AOID as parameters is possible.	
4.	Observe the output of the interface function “ArchiveUpdateResponse”.	A positive feedback is received. No error message or error code occurs. A new Version ID <n> is assigned.	
5.	By using the interface function “ArchiveUpdateRequest” and the AOID from step 2, try to replace the existing data element with an empty element.	The call of the function with this AOID and the empty data element as parameters is possible.	
6.	Observe the output of the interface function “ArchiveUpdateResponse”.	A positive feedback is received. No error message or error code occurs. A new Version ID <n+1> is assigned and returned.	
7.	Using the interface function „ArchiveRetrievalRequest“, the AOID from step 2 and the Version ID from step 6.	The call of the function with this AOID and Version ID as parameters is possible.	
8.	Observe the output of the interface function “ArchiveRetrievalResponse”.	A positive feedback is received. No error message or error code occurs. An XAIP is received.	

9.	Check whether the data element is included and whether this data element is identical to the data element used in step 3.	The data element is not included.	
10.	Using the interface function „ArchiveRetrievalRequest“, the AOID from step 2 and the Version ID from step 4.	The call of the function with this AOID and Version ID as parameters is possible.	
11.	Observe the output of the interface function “ArchiveRetrievalResponse”.	A positive feedback is received. No error message or error code occurs. An XAIP is received.	
12.	Check whether the data element is included and whether this data element is identical to the data element used in step 3.	The data element is included and is identical to the data element used in step 3.	
Verdict			

4.5.4.3.6 S.4.2-10 – All updates are logged

Identifier	S.4.2-10		
Requirement	MD:A5.1-17		
Test Purpose	The test shall verify that all changes are logged to a log file.		
Configuration	CONFIG_ArchiSafe		
Pre-test conditions	• Tester has read permissions on the file system • Test case S.4.2-03 has been performed • If required, perform identification and authentication		
Step	**Test sequence**	**Expected Results**	**Observations**
1.	Check the vendor documentation how and where the middleware records the updates.	A log file exists, the updates are recorded directly within the XAIPs or there is any other type of records, especially for the BINs.	
2.	Check the log records for update events triggered in test case S.4.2-09.	All the updates have been logged, incl. the time when the updates were performed, the changed data and the user name of the person/account who updated the data.	
Verdict			

4.5.4.4 Archive Retrieval Request

4.5.4.4.1 S.4.3-01 – AOID and secure channel is required for retrieval

Identifier	S.4.3-01
Requirement	MD:A5.1-19 MD:A5.1-18 M1:A4.0-5 M1:A4.3-1 M1:A4.3-2
Test Purpose	The test shall verify that the upstream IT applications can send and retrieve any data only through a secure communication channel and only if a valid AOID (if required) is used as a parameter.
Configuration	CONFIG_ArchiSafe
Pre-test conditions	• Tester has read permissions on the Middleware • If required, perform identification and authentication

Step	Test sequence	Expected Results	Observations
1.	Start a data traffic capture tool to monitor the traffic between upstream client application and ArchiSafe.	Data traffic capturing is started.	
2.	Store some XAIP_OKs or BINs using the interface function “ArchiveSubmissionRequest”.	The calls of the function with this XAIP / BIN as a parameter are possible.	
3.	Observe the output of the interface functions “ArchiveSubmissionResponse”.	A positive feedback is received. No error message or error code occurs. An AOID is assigned per stored object.	
4.	Use the interface function “ArchiveRetrievalRequest“ and one AOID from step 2 to request the XAIP.	The call of the function with this AOID as a parameter is possible.	
5.	Observe the output of the interface function “ArchiveRetrievalResponse”.	A positive feedback is received. No error message or error code occurs. An XAIP is received.	
6.	Use several interface functions “ArchiveRetrievalRequest“ and several AOIDs from step 2 to request some XAIPs.	The calls of the function with these AOIDs as a parameter are possible.	
7.	Observe the output of the interface functions “ArchiveRetrievalResponse”.	A positive feedback is received. No error message or error code occurs. All the requested XAIPs are received.	
8.	Use the interface function “ArchiveRetrievalRequest“	The call of the function with this AOID as a parameter is possible.	

	and an AOID which does not exist to request an XAIP.		
9.	Observe the output of the interface function “ArchiveRetrievalResponse”.	A negative feedback will be received. An error message or error code occurs. No XAIP is received.	
10.	If supported, use several “ArchiveUpdateRequest” functions with the AOIDs from step 2 to change the data contained within all the XAIP or XAIP(BIN).	The function calls are possible.	
11.	Check the output of the “ArchiveUpdateResponse” functions.	A new version ID per XAIP / XAIP(BIN) is received.	
12.	Use several interface functions “ArchiveRetrievalRequest“ and several AOIDs from step 2 together with the respective version IDs from step 11 to request some XAIPs.	The calls of the function with these AOIDs as a parameter are possible.	
13.	Observe the output of the interface functions “ArchiveRetrievalResponse”.	A positive feedback is received. No error message or error code occurs. The correct versions of all the requested XAIPs are received.	
14.	If supported, use the “ArchiveUpdateRequest” function with an AOID which does not exist.	The function call is possible.	
15.	Check the output of the “ArchiveUpdateResponse” function.	A negative feedback will be received. An error message or error code occurs.	
16.	Use the interface function “ArchiveRetrievalRequest” using the AOID from step 4 for all versions (e.g. with Version ID = “all”).	The call of the function with this AOID as a parameter is possible.	
17.	Observe the output of the interface function “ArchiveRetrievalResponse”.	A positive feedback is received. No error message or error code occurs. The XAIP with the AOID of step 4 with all versions is received.	
18.	Use“ArchiveEvidenceRequest” function with the AOID from step 4 to check the XAIP / XAIP(BIN) authenticity and integrity for all versions (e.g. with Version ID = “all”).	The function call is possible.	
19.	Check the output of the “ArchiveEvidenceResponse” functions.	For each existing version of this AOID an Evidence Record is received.	
20.	If supported, use the interface function “ArchiveRetrievalRequest” using the AOID from step 4 for all versions (e.g. with Version ID = “all”) demanding also all Evidence Records (e.g. “IncludeERS”)..	The call of the function with this AOID as a parameter is possible.	

21.	Observe the output of the interface function “ArchiveRetrievalResponse”.	A positive feedback is received. No error message or error code occurs. The XAIP with the AOID of step 4 with all versions is received. Furthermore for each Version ID an Evidence Record is received.	
22.	Check the output of the “ArchiveEvidenceResponse” functions.	For each version of the archive data object, one Evidence Record is received. The Evidence Records of this step are equal to the Evidence Records of step 19.	
23.	Use the “ArchiveEvidenceRequest” function with an AOID which does not exist.	The function call is possible.	
24.	Check the output of the “ArchiveEvidenceResponse” function.	A negative feedback will be received. An error message or error code occurs.	
25.	If supported, use the “ArchiveDataRequest” function with one AOID from step 2 and the dataLocation parameter to identify an individual data element within the XAIP / XAIP(BIN).	The function call is possible.	
26.	Check the output of the “ArchiveDataResponse” function.	The requested data value and the original locationValue are received.	
27.	If supported, use the “ArchiveDataRequest” function with an AOID which does not exist.	The function call is possible.	
28.	Check the output of the “ArchiveDataResponse” function.	A negative feedback will be received. An error message or error code occurs.	
29.	Use the “ArchiveDeletionRequest” function with an AOID which does not exist.	The function call is possible.	
30.	Check the output of the “ArchiveDeletionResponse” function.	A negative feedback will be received. An error message or error code occurs.	
31.	Use the “ArchiveDeletionRequest” function with one AOID from step 2 to delete the XAIP / BIN.	The function call is possible.	
32.	Check the output of the “ArchiveDeletionResponse” function.	The XAIP / BIN has been deleted from the storage.	
33.	Stop the data traffic capture tool.	Data traffic capturing is stopped.	
34.	Check the captured data.	The captured data is encrypted or otherwise protected. No references to the previous access procedures can be found.	
Verdict			

4.5.4.4.2 S.4.3-02 – Archive Retrieval returns XAIP

Identifier	S.4.3-02
Requirement	MD:A6.3-2 M1:A4.3-3
Test Purpose	The test shall verify that requested data is always returned in an XAIP-based container.
Configuration	CONFIG_ArchiSafe
Pre-test conditions	• Middleware documentation is available • If required, perform identification and authentication The following steps must be accomplished before starting the test: 1. The call of the function “ArchiveSubmissionRequest” with a XAIP_OK as a parameter is possible. A positive feedback is received. No error message or error code occurs. An AOID is assigned. 2. The call of the function “ArchiveSubmissionRequest” with a XAIP_OK_Sig as a parameter is possible. A positive feedback is received. No error message or error code occurs. An AOID is assigned. 3. The call of the function “ArchiveSubmissionRequest” with a BIN_OK as a parameter is possible. A positive feedback is received. No error message or error code occurs. An AOID is assigned. 4. The call of the function “ArchiveSubmissionRequest” with a BIN_OK_Sig as a parameter is possible. A positive feedback is received. No error message or error code occurs. An AOID is assigned.

Step	Test sequence	Expected Results	Observations
1.	Using the interface function “ArchiveRetrievalRequest“ and the AOID from step 1 in the pre-test conditions to request the XAIP.	The call of the function with this AOID as a parameter is possible.	
2.	Observe the output of the interface function “ArchiveRetrievalResponse”.	A positive feedback is received. No error message or error code occurs. An XAIP is received.	
3.	Using the interface function “ArchiveRetrievalRequest“ and the AOID from step 2 in the pre-test conditions to request the XAIP.	The call of the function with this AOID as a parameter is possible.	
4.	Observe the output of the interface function “ArchiveRetrievalResponse”.	A positive feedback is received. No error message or error code occurs. An XAIP is received.	
5.	Using the interface function “ArchiveRetrievalRequest“ and the AOID from step 3 in the pre-test conditions to request the XAIP.	The call of the function with this AOID as a parameter is possible.	

6.	Observe the output of the interface function “ArchiveRetrievalResponse”.	A positive feedback is received. No error message or error code occurs. An XAIP is received.	
7.	Using the interface function “ArchiveRetrievalRequest“ and the AOID from step 4 in the pre-test conditions to request the XAIP.	The call of the function with this AOID as a parameter is possible.	
8.	Observe the output of the interface function “ArchiveRetrievalResponse”.	A positive feedback is received. No error message or error code occurs. An XAIP is received.	
9.	Check the retrieved XAIPs.	All data objects can successfully be retrieved from the archive system, encapsulated in valid XAIPs as defined in the middleware documentation.	
10.	Check the XML schema of the retrieved XAIPs.	The XML schema of all the XAIPs must comply with an XSD configured by the user or a default XSD of the TOT.	
Verdict			

4.5.4.5 Archive Evidence Request

4.5.4.5.1 S.4.4-01 – Preservation of evidence does not impair possibility to use documents

Identifier	S.4.4-01
Requirement	MD:A6.1-2
Test Purpose	The test shall verify that the procedures used for the preservation of evidence of signed electronic documents do not impair the ability to continue using the electronic documents from the archive.
Configuration	CONFIG_ArchiSafe
Pre-test conditions	• If required, perform identification and authentication • The call of the function “ArchiveSubmissionRequest” with a XAIP_OK _Sig as a parameter is possible. A positive feedback is received. No error message or error code occurs. An AOID is assigned. • The call of the function “ArchiveSubmissionRequest” with a BIN_OK_Sig as a parameter is possible. A positive feedback is received. No error message or error code occurs. An AOID is assigned.

Step	Test sequence	Expected Results	Observations
1.	Start the signature renewal process.	The signature renewal is in process.	
2.	Use the interface function “ArchiveRetrievalRequest“ to request an XAIP.	The call of the function is possible.	
3.	Observe the output of the interface function “ArchiveRetrievalResponse”.	A positive feedback is received. No error message or error code occurs. An XAIP is received.	
4.	Using the interface function “ArchiveRetrievalRequest“ to request the binary object in form of a XAIP.	The call of the function is possible.	
5.	Observe the output of the interface function “ArchiveRetrievalResponse”.	A positive feedback is received. No error message or error code occurs. An XAIP is received.	
6.	Check the retrieved XAIPs and especially the content data.	All data objects can successfully be retrieved from the archive system, encapsulated in valid XAIPs as defined in the middleware documentation. The actual content data is not modified and can be used as usual.	
7.	Start the hash-tree renewal process.	The hash-tree renewal is in process.	
8.	Using the interface function “ArchiveRetrievalRequest“ to request an XAIP.	The call of the function is possible.	

9.	Observe the output of the interface function “ArchiveRetrievalResponse”.	A positive feedback is received. No error message or error code occurs. An XAIP is received.	
10.	Using the interface function “ArchiveRetrievalRequest“ to request the binary object in form of an XAIP.	The call of the function is possible.	
11.	Observe the output of the interface function “ArchiveRetrievalResponse”.	A positive feedback is received. No error message or error code occurs. An XAIP is received.	
12.	Check the retrieved XAIPs and especially the content data.	All data objects can successfully be retrieved from the archive system, encapsulated in valid XAIPs as defined in the middleware documentation. The actual content data is not modified and can be used as usual.	
Verdict			

4.5.4.5.2 S.4.4-02 – Middleware returns correct Evidence Records for each requested AOID

Identifier	S.4.4-02
Requirement	M1:A4.5-1 M1:A4.5-2 M1:A4.5-3
Test Purpose	The test shall verify that requesting Evidence Records for a valid AOID the Evidence Records arecorrect, i. e. conform with ERs specified in RFC 4998 or RFC 6283 and for each Version ID of an AOID there is an Evidence Record. assigned to the AOID.
Configuration	CONFIG_ArchiSafe
Pre-test conditions	• Tester has read/write permissions on the Middleware • If required, perform identification and authentication • Test case M.3-06 has already been successfully checked

Step	Test sequence	Expected Results	Observations
1.	Using several interface functions “ArchiveEvidenceRequest“ with valid AOIDs without Version IDs as parameter.	The calls of the function with an AOID as a parameter are possible.	
2.	Observe the output of the interface function “ArchiveEvidenceResponse”.	A positive feedback is received. No error message or error code occurs. Evidence Records per AOID are received.	
3.	Check the retrieved Evidence Records with an appropriate tool.	There are correct Evidence Records in ERS notation as specified in RFC 4998 or RFC 6283 for the last Version ID of each XAIP/AOID or BIN/AOID. The AOIDs are exactly these AOIDs passed over as parameters.	
4.	Using several interface functions “ArchiveEvidenceRequest“ with valid AOIDs for all versions (e.g. Version ID =”all”) as parameters.	The calls of the function with these AOIDs and Version IDs as parameters are possible.	
5.	Observe the output of the interface function “ArchiveEvidenceResponse”.	A positive feedback is received. No error message or error code occurs. Evidence Records per AOID are received.	
6.	Check the retrieved Evidence Records with an appropriate tool.	There is a correct Evidence Record in ERS notation as specified in RFC 4998 or RFC 6283 for each Version ID of each XAIP/AOID or BIN/AOID. The AOIDs are exactly these AOIDs passed over as parameters.	
7.	Using the interface function “ArchiveEvidenceRequest“	The call of the function with one AOID as a parameter is possible.	

	with one valid AOID and one valid Version ID as parameters in one function call.		
8.	Observe the output of the interface function “ArchiveEvidenceResponse”.	A positive feedback is received. No error message or error code occurs. An Evidence Record is received.	
9.	Check the retrieved Evidence Record by an appropriate tool.	There is a correct Evidence Record in ERS notation as specified in RFC 4998 or RFC 6283 and contains one Evidence Records in ERS notation associated to the valid Version ID of the valid AOID of step 7.. The AOID and Version ID are exactly the AOID and Version ID passed over as parameter. The tool shows that the ERs are formed correctly.	
10.	Use the interface function “ArchiveEvidenceRequest“ and an AOID which does not exist to request an Evidence Record.	The call of the function with this AOID as a parameter is possible.	
11.	Observe the output of the interface function “ArchiveEvidenceResponse”.	A negative feedback is received. An error message or error code occurs. No Evidence Record is received.	
12.	Use the interface function “ArchiveEvidenceRequest“ and an existing AOID and a Version ID which does not exist to request an Evidence Record.	The call of the function with this AOID as a parameter is possible.	
13.	Observe the output of the interface function “ArchiveEvidenceResponse”.	A negative feedback is received. An error message or error code occurs. No Evidence Record is received.	
Verdict			

4.5.4.5.3 S.4.4-03 – Middleware creates correct Evidence Records for specific XAIP or BIN versions

Identifier	S.4.4-03
Requirement	MD:A5.1-24 M1:A4.5-4
Test Purpose	The test shall verify that the middleware is able to create correct electronic Evidence Records for each version of an XAIP or BIN so that their authenticity and integrity since the time of archiving is ensured even if changes were performed in the meantime.
Configuration	CONFIG_ArchiSafe
Pre-test conditions	• Tester has read/write permissions on the Middleware • If required, perform identification and authentication • The call of the function “ArchiveSubmissionRequest” with a XAIP_OK_Sig as a parameter is possible. A positive feedback is received. No error message or error code occurs. An AOID A1 is assigned. • The call of the function “ArchiveUpdateRequest” with a valid AOID and for adding a DXAIP_OKas a parameter is possible. A positive feedback is received. No error message or error code occurs. A new Version ID is received.

Step	Test sequence	Expected Results	Observations
1.	Using the interface function “ArchiveEvidenceRequest“ and a valid AOID to request the Evidence Records for the XAIP/BIN.	The call of the function with this AOID as parameter is possible	
2.	Observe and check the output of the interface function “ArchiveEvidenceResponse” with an appropriate tool.	A positive feedback is received. No error message or error code occurs. The correct Evidence Records in ERS as specified in RFC 4998 or RFC 6283 is received.	
3.	Using the interface function “ArchiveEvidenceRequest“ with a valid AOID and an assigned version ID indicating the very first version to request the Evidence Record for the archived XAIP/BIN.	The call of the function with this AOID and the Version ID as parameters is possible.	
4.	Observe and check the output of the interface function “ArchiveEvidenceResponse” with an appropriate tool.	A positive feedback will be received; no error message or error code. A correct Evidence Record in ERS as specified in RFC 4998 or RFC 6283 is received.	
5.	Evaluate the received Evidence Records from step 2 and 4 by using an appropriate tool.	The Evidence Records are valid with respect to specification in RFC 4998 or RFC 6283, and contain the necessary data to prove the integrity and authenticity of the XAIP versions. The hash values of the Evidence Records from step 4 and one Evidence Record of step 2 are equal and cover therefore the same version of the XAIP/BIN.	

		In step 4 there is one Evidence Record for one Version ID. In step 2 for each Version ID of the AOID, there is one Evidence Record which contains evidences for this version of the XAIP/BIN. The integrity and authenticity can be proven back to the time of first archival.	
Verdict			

4.5.4.6 Archive Deletion Request

4.5.4.6.1 S.4.5-01 – Deletion is only possible by authorised entities and with included reason

Identifier	S.4.5-01
Requirement	MD:A5.1-28 MD:A5.1-27 M1:A4.4-3 M1:A5.0-3
Test Purpose	The test shall verify that deletion of data before their expiry date can only be performed by authorised users of an authorised IT application when the reason for deletion is contained in the deletion request.
Configuration	CONFIG_ArchiSafe
Pre-test conditions	• Tester has read/write permissions on the middleware • Authentication against the application with the credentials of a user who is authorised to access that just submitted XAIP/BIN but not authorised to delete data before it is expired, is successfully. • The call of the function “ArchiveSubmissionRequest” with a XAIP_OK_Sig or BIN_OK_Sig as a parameter is possible. A positive feedback is received. No error message or error code occurs. An AOID is assigned.

Step	Test sequence	Expected Results	Observations
1.	Using the interface function “ArchiveDeletionRequest“ and a valid AOID to request the deletion of an archived XAIP_OK_SIG or BIN. Do not provide a reason for deletion.	The call of the function with this AOID as a parameter is possible.	
2.	Observe the output of the interface function “ArchiveDeletionResponse”.	A negative feedback is received. An error message or error code occurs. The XAIP / BIN is not deleted.	
3.	Using the interface function “ArchiveDeletionRequest“ and the AOID to request the deletion of the archived XAIP_OK_SIGorBIN . Provide a reason for deletion.	The call of the function with this AOID as a parameter is possible.	
4.	Observe the output of the interface function “ArchiveDeletionResponse”.	A negative feedback is received. An error message or error code occurs. The XAIP / BIN is not deleted.	
5.	Authenticate against the application with the credentials of a user who is authorised not only to access the XAIP submitted but also to delete data before it is expired.	The user has been authenticated successfully.	
6.	Using the interface function “ArchiveDeletionRequest“ and the AOID to request the deletion of the	The call of the function with this AOID as a parameter is possible.	

	XAIP_OK_SIG or BIN . Do not provide a reason for deletion.		
7.	Observe the output of the interface function “ArchiveDeletionResponse”.	A negative feedback is received. An error message or error code occurs. The XAIP / BIN is not deleted.	
8.	Using the interface function “ArchiveDeletionRequest“ and the AOID to request the deletion of the XAIP_OK_SIG or BIN . Provide a reason for deletion.	The call of the function with this AOID as a parameter is possible.	
9.	Observe the output of the interface function “ArchiveDeletionResponse”.	A positive feedback is received. No error message or error code occurs. The XAIP / BIN is deleted.	
Verdict			

4.5.4.6.2 S.4.5-02 – Deletion shall be performed for complete XAIP / BIN

Identifier	S.4.5-02
Requirement	MD:A5.1-29
Test Purpose	The test shall verify that a deletion is always performed for the complete XAIP / BIN, including all versions of data objects.
Configuration	CONFIG_ArchiSafe
Pre-test conditions	• Tester has read/write permissions on the middleware • If required, perform identification and authentication

Step	Test sequence	Expected Results	Observations
1.	Store an XAIP_OK or BIN using the interface function “ArchiveSubmissionRequest”.	The call of the function with this XAIP / BIN as a parameter is possible.	
2.	Observe the output of the interface function “ArchiveSubmissionResponse”.	A positive feedback is received. No error message or error code occurs. An AOID is assigned.	
3.	Using the interface function “ArchiveUpdateRequest” and the AOID from step 2 add a DXAIP_OK to the previously stored XAIP_OK / XAIP(BIN) .	The call of the function with this DXAIP_OK and the AOID as parameters is possible.	
4.	Observe the output of the interface function “ArchiveUpdateResponse”.	A positive feedback is received. No error message or error code occurs. A new Version ID is received.	
5.	Using the interface function “ArchiveUpdateRequest” and the AOID from step 2, to change the XAIP_OK or / XAIP(BIN)(e.g. changing metadata).	The call of the function with this XAIP and the AOID as parameters is possible.	
6.	Observe the output of the interface function “ArchiveUpdateResponse”.	A positive feedback is received. No error message or error code occurs. A new Version ID is received.	
7.	Using the interface function “ArchiveDeletionRequest” and the AOID from step 2 to delete the XAIP_OK / BIN.	The call of the function with this AOID as a parameter is possible.	
8.	Observe the output of the interface function “ArchiveDeletionResponse”.	A positive feedback is received. No error message or error code occurs. The XAIP/BIN is deleted.	
9.	Try to retrieve an earlier version of the XAIP / BIN by using an “ArchiveRetrievalRequest” with the AOID from step 2 without a Version ID and with all possible and valid version ID's (see steps 4 and 6).	The call of the function is possible.	

10.	Observe the output of the interface function “ArchiveRetrievalResponse”.	A negative feedback will be received. An error message or error code occurs. No XAIP/BIN is retrieved in any case.	
Verdict			

4.5.4.6.3 S.4.5-03 – Deletion requires reason, expiration and AOID

Identifier	S.4.5-03
Requirement	MD:A5.1-28 MD:A5.1-27 M1:A4.4-4 M1:A4.4-6
Test Purpose	The test shall verify that an “ArchiveDeletionRequest” will not delete an XAIP/BIN before its expiration, if the AOID is invalid or there is no reason given for the deletion and that the log file will always log the deletion including the reason.
Configuration	CONFIG_ArchiSafe
Pre-test conditions	• Tester has read/write permissions on the middleware • Tests S.4.5-01 and S.4.5-03 have been performed successfully • If required, perform identification and authentication

Step	Test sequence	Expected Results	Observations
1.	Store an XAIP_OK_SIG or BIN using the interface function “ArchiveSubmissionRequest”.	The call of the function with this XAIP / BIN as a parameter is possible.	
2.	Observe the output of the interface function “ArchiveSubmissionResponse”.	A positive feedback is received. No error message or error code occurs. An AOID is assigned.	
3.	Using the interface function “ArchiveDeletionRequest“ and the AOID from step 2 to request the deletion of the XAIP_OK_SIG or BIN. Do not provide a reason for deletion.	The call of the function with this AOID as a parameter is possible.	
4.	Observe the output of the interface function “ArchiveDeletionResponse”.	A negative feedback is received. An error message or error code occurs. The XAIP / BIN is not deleted.	
5.	Using the interface function “ArchiveDeletionRequest“ and an invalid AOID request the deletion of an XAIP or BIN. Provide a reason for deletion.	The call of the function with this AOID as a parameter is possible.	
6.	Observe the output of the interface function “ArchiveDeletionResponse”.	A negative feedback is received. An error message or error code occurs. No XAIP / BIN is deleted.	
7.	Using the interface function “ArchiveDeletionRequest“ and the AOID from step 2 request the deletion of the XAIP_OK_SIG or BIN. Provide a reason for deletion.	The call of the function with this AOID as a parameter is possible.	

8.	Observe the output of the interface function “ArchiveDeletionResponse”.	A positive feedback is received. No error message or error code occurs. The XAIP / BIN is deleted.	
9.	Check the log file for the deletion procedure.	The log file contains all the data about the deletion of this XAIP / BIN including the reason for deletion.	
Verdict			

4.5.4.6.4 S.4.5-04 – Deletion of an archive object shall be logged

Identifier	S.4.5-04
Requirement	MD:A5.1-5 MD:A5.1-31
Test Purpose	The test shall verify that every deletion is logged.
Configuration	CONFIG_ArchiSafe
Pre-test conditions	• Tester has read/write permissions on the middleware. • If required, perform identification and authentication.

Step	Test sequence	Expected Results	Observations
1.	Check for the existence of a log file or any other type of records that is used by the middleware to log deletions.	There is such an event log.	
2.	Store an XAIP_OK_SIG or BIN using the interface function “ArchiveSubmissionRequest”.	The call of the function with this XAIP / BIN as a parameter is possible.	
3.	Observe the output of the interface function “ArchiveSubmissionResponse”.	A positive feedback is received. No error message or error code occurs. An AOID is assigned.	
4.	Using the interface function “ArchiveDeletionRequest” and the AOID from step 3, delete the XAIP_OK_SIG or theBIN with a reason for deletion.	The call of the function with this AOID as a parameter is possible.	
5.	Observe the output of the interface function “ArchiveDeletionResponse”.	A positive feedback is received. No error message or error code occurs. The XAIP / BIN is deleted.	
6.	Check the log for the log data of the deletion procedure.	The log contains all the data about the deletion of the XAIP / BIN including the reason why it was deleted.	
Verdict			

4.5.4.6.5 S.4.5-05 – Error message if deletion is not supported

Identifier	S.4.5-05
Requirement	M1:A4.4-2
Test Purpose	The test shall verify that the ArchiSafe module replies to an “ArchiveDeletionRequest” with an error message if the ECM/long-term storage has no deletion function or the used storage media does not allow deletion.
Configuration	CONFIG_ArchiSafe
Pre-test conditions	• Tester has read/write/delete permissions • The user manual for the ECM/Long-term storage is available • A storage system which supports deletion and a storage system which doesnt support deletion are present.

Step	Test sequence	Expected Results	Observations
1.	Use a storage for the test which supports deletion.	---	
2.	Store an XAIP_OK_SIG or BIN using the interface function “ArchiveSubmissionRequest”.	The call of the function with this XAIP / BINas a parameter is possible.	
3.	Observe the output of the interface function “ArchiveSubmissionResponse”.	A positive feedback is received. No error message or error code occurs. An AOID is assigned.	
4.	Using the interface function “ArchiveDeletionRequest“[12] and the AOID from step 2 request the deletion of the XAIP_OK_SIG or BIN of step 2.	The call of the function with this AOID as a parameter is possible.	
5.	Observe the output of the interface function “ArchiveDeletionResponse”.	A positive feedback is received. No error message or error code occurs. The XAIP / BIN is deleted.	
6.	Use a storage for the test which does not support deletion.	---	
7.	Store an XAIP_OK_SIG or BIN using the interface function “ArchiveSubmissionRequest”.	The call of the function with this XAIP / BIN as a parameter is possible.	
8.	Observe the output of the interface function “ArchiveSubmissionResponse”.	A positive feedback is received. No error message or error code occurs. An AOID is assigned.	
9.	Using the interface function “ArchiveDeletionRequest“	The call of the function with this AOID as a parameter is possible.	

[12] The XAIP or BIN has become an expired XAIP or XAIP(BIN).

	and the AOID from step 8 to delete the XAIP / BIN[13].		
10.	Observe the output of the interface function “ArchiveDeletionResponse”.	An error message or error code is received.	
Verdict			

[13] The XAIP or BIN has become an expired XAIP or XAIP(BIN).

4.5.4.6.6 S.4.5-06 – Deletion should be possible in an irreversible manner

Identifier	S.4.5-06
Requirement	M1:A4.4-5
Test Purpose	The test shall verify that the ArchiSafe module is able to initiate a permanent deletion of XAIPs/BINs in the ECM/long-term storage.
Configuration	CONFIG_ArchiSafe
Pre-test conditions	• Tester has administration permissions on the file system • The middleware user manual is available • The user manual for the ECM/Long-term storage is available • The ECM/Long-term storage supports permanent deletion • Check the ArchiSafe documentation how the permanent deletion in the storage can be configured/initiated. • Configure ArchiSafe and the storage in such a way that the permanent deletion will be used.

Step	**Test sequence**	**Expected Results**	**Observations**
1.	Store an XAIP_OK or BIN using the interface function “ArchiveSubmissionRequest”.	The call of the function with this XAIP / BIN as a parameter is possible.	
2.	Observe the output of the interface function “ArchiveSubmissionResponse”.	A positive feedback is received. No error message or error code occurs. An AOID is assigned.	
3.	Using the interface function “ArchiveDeletionRequest“ and the AOID from step 2 to request the deletion of the XAIP_OK / BIN.	The call of the function with this AOID as a parameter is possible.	
4.	Observe the output of the interface function “ArchiveDeletionResponse”.	A positive feedback is received. No error message or error code occurs. The XAIP / BIN is deleted.	
5.	Use all available (administration) functions of ArchiSafe and the storage for attempting to recover the XAIP.	The deleted XAIPs / BINs cannot be recovered.	
Verdict			

4.5.4.7 Archive Data Request

Pre-supposition:

A product which claims to to comply with the “ArchiveDataRequest/-Response” - functionality according to M.1-04 “ArchiveDataRequest” of this TR has to pass the following test case or prove that it supports functional analogous functions.

4.5.4.7.1 S.4.6-01 – Archive Data Request shall require valid AOID and dataLocation

Identifier	S.4.6-01
Requirement	M1:A4.6-1 M1:A4.6-2
Test Purpose	The test shall verify that the “ArchiveDataRequest” will retrieve and return a data element from an XAIP/BIN, if the request is performed with a valid AOID and at least one valid dataLocation parameter. The test shall verify that data elements that are retrieved with an “ArchiveDataRequest” are returned as they have been stored originally without being changed. The test shall verify that an “ArchiveDataRequest” with an invalid AOID returns an understandable error code or error message.
Configuration	CONFIG_ArchiSafe
Pre-test conditions	• If required, perform identification and authentication

Step	Test sequence	Expected Results	Observations
1.	Store several XAIP_OK's or BIN's using the interface functions „ArchiveSubmissionRequest“.	The calls of the function with this XAIP as a parameter are possible.	
2.	Observe the output of the interface functions “ArchiveSubmissionResponse”.	Positive feedbacks are received. No error messages or error codes arereturned. A list of AOIDs has been assigned.	
3.	If the interface function “ArchiveDataRequest” is implemented, use the interface function “ArchiveDataRequest” with one AOID from step 2 with one valid dataLocation parameter to retrieve a data element that has been stored in the XAIP_OK / BIN in step 1.	The call of the function with these parameters is possible.	
4.	Observe the output of the interface function “ArchiveDataResponse”.	A positive feedback is received. No error message or error code is returned. The intended data element is received.	
5.	Compare the retrieved data element with the version that has originally been stored in the XAIP / BIN in step 1.	The data elements are equal.	

6.	Use the interface function “ArchiveDataRequest” with all the AOIDs from step 2 with one valid dataLocation parameter to retrieve the data elements that has been stored in the XAIP_OK's / BIN's in step 1.	The call of the function with these parameters is not possible at all or an error occurs.	
7.	If the interface function “ArchiveDataRequest” is implemented, use the interface function “ArchiveDataRequest” with one AOID from step 2 with two valid dataLocation parameters to retrieve a data element that has been stored in the XAIP_OK / BIN in step 1.	The call of the function with these parameters is possible.	
8.	Observe the output of the interface function “ArchiveDataResponse”.	A positive feedback is received. No error message or error code is returned. The data element's of the addressed XAIP / BIN are received.	
9.	Compare the retrieved data element with the version that has originally been stored in the XAIP / BIN in step 1.	The data elements are equal.	
10.	If the interface function “ArchiveDataRequest”, use the interface function “ArchiveDataRequest” with an invalid AOID and an arbitrary dataLocation parameter.	The call of the function with these parameters is possible.	
11.	Observe the output of the interface function “ArchiveDataResponse”.	A negative feedback is received. An error message or error code is returned. No data element is received.	
12.	If the interface function “ArchiveDataRequest”, use the interface function “ArchiveDataRequest” with one AOID from step 2 with an invalid dataLocation parameter.	The call of the function with these parameters is possible.	
13.	Observe the output of the interface function “ArchiveDataResponse”.	A negative feedback is received. An error message or error code is returned. No data element is received.	
Verdict			

4.5.4.7.2 S.4.7-01 – ArchiSafe Module is robust against incorrect parameters

Identifier	S.4.7-01
Requirement	M1:A4.0-2
Test Purpose	The test shall verify that the ArchiSafe Module's functionality is not negatively affected by false or incorrectly parametrised requests. Note: Keep in mind to skip any step which will not supported by the TOT, especially regarding the "ArchiveSubmissionRequests"
Configuration	CONFIG_ArchiSafe
Pre-test conditions	• If required, perform identification and authentication • Developer documentation is available, which contains information about existing restrictions for the length and admissible characters of an AOID

Step	Test sequence	Expected Results	Observations
1.	Use the interface function "ArchiveSubmissionRequest" with no parameters.	The request is answered with a clear and understandable error message or an error code.	
2.	Use the interface function "ArchiveSubmissionRequest" with a binary data object with 0 bytes length.	The request is performed correctly. An AOID is returned. The object can be retrieved without errors and modifications.	
3.	Use the interface function "ArchiveSubmissionRequest" with a very large archive object (several Gigabytes, at least four).	The request is performed correctly. An AOID is returned. The object can be retrieved without errors and modifications.	
4.	Use the interface function "ArchiveSubmissionRequest" with an archive object which contains nested XAIPs (at least 5 levels).	The request is performed correctly. An AOID is returned. The object can be retrieved without errors and modifications.	
5.	Use the interface function "ArchiveUpdateRequest" with no parameters.	The request is answered with a clear and understandable error message or an error code.	
6.	Use the interface function "ArchiveUpdateRequest" with an AOID that contains invalid characters.	The request is answered with a clear and understandable error message or an error code.	
7.	Use the interface function "ArchiveUpdateRequest" with an AOID that contains too many characters.	The request is answered with a clear and understandable error message or an error code.	
8.	Use the interface function "ArchiveUpdateRequest" and try to update elements and sections of an archived XAIP which do not exist yet.	The update will be performed. The elements and sections will added only to the XAIP. Existing elements/sections will not be modified.	
9.	Use the interface function "ArchiveRetrievalRequest" with no parameters.	The request is answered with a clear and understandable error message or an error code.	

10.	Use the interface function “ArchiveRetrievalRequest” with an AOID that contains invalid characters.	The request is answered with a clear and understandable error message or an error code.	
11.	Use the interface function “ArchiveRetrievalRequest” with an AOID that contains too many characters.	The request is answered with a clear and understandable error message or an error code.	
12.	Use the interface function “ArchiveRetrievalRequest” with a version ID that contains invalid characters.	The request is answered with a clear and understandable error message or an error code.	
13.	Use the interface function “ArchiveRetrievalRequest” with a version ID that contains too many characters.	The request is answered with a clear and understandable error message or an error code.	
14.	Use the interface function “ArchiveEvidenceRequest” with no parameters.	The request is answered with a clear and understandable error message or an error code.	
15.	Use the interface function “ArchiveEvidenceRequest” with an AOID that contains invalid characters.	The request is answered with a clear and understandable error message or an error code.	
16.	Use the interface function “ArchiveEvidenceRequest” with an AOID that contains too many characters.	The request is answered with a clear and understandable error message or an error code.	
17.	Use the interface function “ArchiveEvidenceRequest” with a version ID that contains invalid characters.	The request is answered with a clear and understandable error message or an error code.	
18.	Use the interface function “ArchiveEvidenceRequest” with a version ID that contains too many characters.	The request is answered with a clear and understandable error message or an error code.	
19.	Use the interface function “ArchiveDataRequest” with an AOID that contains invalid characters.	The request is answered with a clear and understandable error message or an error code.	
20.	Use the interface function “ArchiveDataRequest” with an AOID that contains too many characters.	The request is answered with a clear and understandable error message or an error code.	
21.	Use the interface function “ArchiveDataRequest” with a valid AOID and a dataLocation parameter that contains invalid characters.	The request is answered with a clear and understandable error message or an error code.	
22.	Use the interface function “ArchiveDataRequest” with a valid AOID and a dataLocation parameter that contains too many characters.	The request is answered with a clear and understandable error message or an error code.	
23.	Use the interface function “ArchiveDeletionRequest” with no parameters.	The request is answered with a clear and understandable error message or an error code.	
24.	Use the interface function “ArchiveDeletionRequest” with an AOID that contains invalid characters.	The request is answered with a clear and understandable error message or an error code.	

25.	Use the interface function “ArchiveDeletionRequest” with an AOID that contains too many characters.	The request is answered with a clear and understandable error message or an error code.	
26.	Use the interface function “ArchiveDeletionRequest” with an AOID that contains wild card characters like “*” or “?”.	The request is answered with a clear and understandable error message or an error code.	
Verdict			

4.5.4.7.3 S.4.8-01 Performance Requirements

Identifier	S.4.8-01
Requirement	There is actually no requirement in the TR, but the TOT shall ensure a suitable performance while executing Archive Requests
Test Purpose	The test shall verify that the TOT is able to ensure a suitable performance while executing Archive Requests.
Configuration	CONFIG_ArchiSafe
Pre-test conditions	• The middleware documentation /user manual is available • The documentation / user manual for the ECM/Long-term storage is available

Step	Test sequence	Expected Results	Observations
1.	Check the documentation of the TOT and (optionally) of the ECM/long-term storage, if there are any assertions and related conditions or constraints regarding the performance of the TOT while executing Archive Requests (means for example: how long does proceeding of a request with an archive object of the size x take)	The documentation of ArchiSafe and (optional) of the ECM/long-term storage contain some assertions and related conditions or constraints regarding the performance of the TOT while executing Archive Requests	
2.	Store an XAIP_OK or BIN_OK using the interface function “ArchiveSubmissionRequest” and measure the assured performance for executing the request, i.e. the time the “ArchiveSubmissionRequest” will be answered by an “ArchiveSubmissionResponse”. Please take care to just measure the TOT performance, not other modules/systems.	The measure confirms the assured performance	
3.	Store an XAIP_OK_Sig or BIN_OK_Sig using the interface function “ArchiveSubmissionRequest” and measure the assured performance to execute the request, i.e. the time the “ArchiveSubmissionRequest” will be answered by an “ArchiveSubmissionResponse”. Please take care to just measure the TOT performance, not other modules/systems.	The measure confirms the assured performance	
4.	Repeat steps 2 and 3 at least with 3 data objects which differ notably in the size.	The measure confirms the assured performance	
5.	Use the AOID retrieved in step 2 for calling an “ArchiveRetrievalRequest” for the retrieval of the corresponding XAIP_OK / BIN_OK and measure the assured performance to execute the request, i. e. measure	The measure confirms the assured performance	

	the time the “ArchiveRetrievalRequest” will be answered by an “ArchiveRetrievalResponse”. Please take care to just measure the TOT performance, not other modules/systems.		
6.	Use the AOID retrieved in step for calling an “ArchiveRetrievalRequest” for the retrieval of the corresponding XAIP_OK_Sig or BIN_OK_Sig and measure the assured performance to execute the request, i. e. measure the time the “ArchiveRetrievalRequest” will be answered by an “ArchiveRetrievalResponse”. Please take care to just measure the TOT performance, not other modules/systems.	The measure confirms the assured performance	
7.	Repeat steps 5 and 6 with the AOID's retrieved in step 4	The measure confirms the assured performance	
Verdict			

4.5.4.8 Verify Request

The test cases of the “VerifyRequest” - function of the interface S.1 (sec. 4.5.1.1 VerifyRequest) are also relevant here.

4.5.4.8.1 S.4.9-01 Verify Request – Verification of signature includes certificate path validation and Evidence Records

Identifier	S.1.1-01
Requirement	M2:A5.1-8 M2:A5.1-9
Test Purpose	The function is able to verify whether the user certificate used to generate the signature was valid at the time the signature was generated (see Chapter 5.1.3). Validity verification shall be complete, i.e. it includes the entire certificate chain back to a trustworthy root certificate. The Cryptographic Module shall be able to verify advanced and qualified electronic signatures. Qualified time stamps with (qualified) electronic signatures as well as Evidence Records as DXAIPs shall be verifiable, i.e. the validity of the time stamp signature at the time of time stamp generation must be verified.
Configuration	CONFIG_Common
Pre-test conditions	• An XAIP_OK_Sig_Q / BIN_OK_Sig_Q is present. XAIP_OK_Sig_Q / BIN_OK_Sig_Q is a XAIP_OK_SIG / BIN with qualified electronic signature • An XAIP_OK_Sig_A / BIN_OK_Sig_A is present. XAIP_OK_Sig_A / BIN_OK_Sig_A is a XAIP_OK_SIG / BIN with advanced electronic signature • An XAIP_OK_Sig_Q_ERS is present. XAIP_OK_Sig_Q_ERS is a XAIP_OK_SIG_OK_ER with qualified electronic signature and at least one evidence record • An XAIP_OK_Sig_A_ERS is present. XAIP_OK_Sig_A_ERS is a XAIP_OK_SIG_OK_ER with advanced electronic signature and at least one evidence record • An XAIP_NOK_Sig_Q / BIN_NOK_Sig_Q is present. XAIP_NOK_Sig_Q / BIN_NOK_Sig_Q is a XAIP_NOK_SIG / BIN_NOK_SIG with qualified electronic signature • An XAIP_NOK_Sig_A / BIN_NOK_Sig_A is present. XAIP_NOK_Sig_A / BIN_NOK_Sig_A is a XAIP_NOK_SIG / BIN_NOK_SIG with advanced electronic signature • An XAIP_NOK_Sig_Q_ERS is present. XAIP_NOK_Sig_Q_ERS is a XAIP_NOK_SIG_OK_ER with qualified electronic signature and at least one evidence record • An XAIP_NOK_Sig_A_ERS is present. XAIP_NOK_Sig_A_ERS is a XAIP_NOK_SIG_OK_ER with advanced electronic signature and at least one evidence record • An XAIP_NOK_ERS is present. XAIP_NOK_ERS is a XAIP_NOK_ER with qualified electronic signature and at least one evidence record • developer documents are present • A DXAIP_OK_SIG is present. DXAIP_OK_SIG is a DXAIP_OK_SIG with qualified or advanced electronic signature referenced to an XAIP_OK • A DXAIP_NOK_SIG is present. DXAIP_NOK_SIG is a DXAIP_NOK_SIG with qualified or advanced electronic signature referenced to an XAIP_OK • developer documents are present • if the Cryptographic Module isn't a certified signature product (e. g. according to BSI-TR-03112) a suitable test-bed should be used to verify the

	correctness of the implementation of the signature-related functionality.		
Step	**Test sequence**	**Expected Results**	**Observations**
1.	Transfer the archival information package XAIP_OK_Sig_Q / BIN_OK_Sig_Q (see pre-test conditions) to the TOT using the interface function “VerifyRequest”.	The call of the function with this XAIP / BIN as parameter is possible.	
2.	Observe the output of the interface function “VerifyResponse”.	A positive feedback will be received; no error message or error code. A VerificationReport is included in “VerifyResponse”.	
3.	Examine the VerificationReport if the validity verification would be done by the Cryptographic Module.	The validity verification shall be correct and complete, i.e. it includes the entire certificate chain back to a trustworthy root certificate.	
4.	Transfer the archival information package XAIP_OK_Sig_A / BIN_OK_Sig_A (see pre-test conditions) to the TOT. using the interface function “VerifyRequest”.	The call of the function with this XAIP / BIN as parameter is possible.	
5.	Observe the output of the interface function “VerifyResponse”.	A positive feedback will be received; no error message or error code. A VerificationReport is included in “VerifyResponse”.	
6.	Examine the VerificationReport if the validity verification would be done by the Cryptographic Module.	The validity verification shall be correct and complete, i.e. it includes the entire certificate chain back to a trustworthy root certificate.	
7.	Transfer the archival information package XAIP_NOK_Sig_Q / BIN_NOK_Sig_Q (see pre-test conditions) to the TOT using the interface function “VerifyRequest” asking for a verification report.	The call of the function with this XAIP / BIN as parameter is possible.	
8.	Observe the output of the interface function “VerifyResponse”.	A negative feedback will be received with error message and error code. A VerificationReport is included in “VerifyResponse”.	
9.	Examine the VerificationReport if the validity verification would be done by the Cryptographic Module.	The Verification Reports includes verification report structures for the signatures, Evidence Records and the XAIP.	
10.	Transfer the archival information package XAIP_NOK_Sig_A / BIN_NOK_Sig_A (see pre-test conditions) to the TOT. using the interface function “VerifyRequest” asking for a verification report.	The call of the function with this XAIP / BIN as parameter is possible.	
11.	Observe the output of the interface function “VerifyResponse”.	A negative feedback will be received with error message and error code. A VerificationReport is included in “VerifyResponse”.	

12.	Examine the VerificationReport if the validity verification would be done by the Cryptographic Module.	The Verification Reports includes verification report structures for the signatures, Evidence Records and the XAIP.	
13.	Transfer the archival information package XAIP_OK_Sig_Q_ERS (see pre-test conditions) to the TOT using the interface function “VerifyRequest”.	The call of the function with this XAIP as parameter is possible.	
14.	Observe the output of the interface function “VerifyResponse”.	A positive feedback will be received; no error message or error code. A VerificationReport is included in “VerifyResponse”. The verification of the ER was also successful.	
15.	Examine the VerificationReport if the validity verification would be done by the Cryptographic Module.	The validity verification shall be correct and complete, i.e. it includes the entire certificate chain back to a trustworthy root certificate.	
16.	Transfer the archival information package XAIP_OK_Sig_A_ERS (see pre-test conditions) to the TOT. using the interface function “VerifyRequest”.	The call of the function with this XAIP as parameter is possible.	
17.	Observe the output of the interface function “VerifyResponse”.	A positive feedback will be received; no error message or error code. A VerificationReport is included in “VerifyResponse”. The verification of the ER was also successful.	
18.	Examine the VerificationReport if the validity verification would be done by the Cryptographic Module.	The validity verification shall be correct and complete, i.e. it includes the entire certificate chain back to a trustworthy root certificate.	
19.	Transfer the archival information package XAIP_NOK_Sig_Q_ERS (see pre-test conditions) to the TOT using the interface function “VerifyRequest” asking for a verification report..	The call of the function with this XAIP as parameter is possible.	
20.	Observe the output of the interface function “VerifyResponse”.	A negative feedback will be received with error message and error code. A VerificationReport is included in “VerifyResponse”.	
21.	Examine the VerificationReport if the validity verification would be done by the Cryptographic Module.	The Verification Reports includes verification report structures for the signatures, Evidence Records and the XAIP.	
22.	Transfer the archival information package XAIP_NOK_Sig_A_ERS (see pre-test conditions) to the TOT using the interface function “VerifyRequest” asking for a verification report.	The call of the function with this XAIP as parameter is possible.	
23.	Observe the output of the interface function “VerifyResponse”.	A negative feedback will be received with error message and error code. A VerificationReport is included in “VerifyResponse”.	

24.	Examine the VerificationReport if the validity verification would be done by the Cryptographic Module.	The Verification Reports includes verification report structures for the signatures, Evidence Records and the XAIP.	
25.	Transfer the archival information package XAIP_NOK_ERS (see pre-test conditions) to the TOT using the interface function “VerifyRequest” asking for a verification report..	The call of the function with this XAIP as parameter is possible.	
26.	Observe the output of the interface function “VerifyResponse”.	A negative feedback will be received with error message and error code. A VerificationReport is included in “VerifyResponse”.	
27.	Examine the VerificationReport if the validity verification would be done by the Cryptographic Module.	The Verification Reports includes verification report structures for the signatures, Evidence Records and the XAIP.	
28.	Transfer the archival information package DXAIP_OK_SIG (see pre-test conditions) to the TOT. using the interface function “VerifyRequest”.	The call of the function with this DXAIP_OK_SIG as parameter is possible.	
29.	Observe the output of the interface function “VerifyResponse”.	A positive feedback will be received; no error message or error code. A VerificationReport is included in “VerifyResponse”. The verification of the DXAIP_OK_SIG was also successful.	
30.	Examine the VerificationReport if the validity verification would be done by the Cryptographic Module.	The validity verification shall be correct and complete, i.e. it includes the entire certificate chain back to a trustworthy root certificate.	
31.	Transfer the archival information package DXAIP_NOK_SIG (see pre-test conditions) to the TOT using the interface function “VerifyRequest” asking for a verification report..	The call of the function with this DXAIP_NOK_OK as parameter is possible.	
32.	Observe the output of the interface function “VerifyResponse”.	A negative feedback will be received with error message and error code. A VerificationReport is included in “VerifyResponse”.	
33.	Examine the VerificationReport if the validity verification would be done by the Cryptographic Module.	The Verification Reports includes verification report structures for the signatures, Evidence Records and the XAIP.	
Verdict			

4.5.5 Interface S.5

The TR-ESOR-S.5 interface enables accesses from the ArchiSafe module to the ECM/long-term storage without technical dependence of the cryptographically secured Evidence Records.

This is an interface of a component not part of the TR-ESOR middleware. Therefore, no conformity tests can be specified here.

4.5.6 Interface S.6

The archiving of (new) archival information packages is possible with the TR-ESOR-S.6 interface described here, which can be used to include the ArchiSig-Module directly in the archiving procedure. This is a direct way to generate the securing hash values. Thus, it is impossible to circumvent this security function.

Pre-supposition:

A product which claims to functionally comply with the Interface S.6 specification of this TR has to pass all test cases in this section or prove that it supports functional analogous interfaces.

4.5.6.1 Archive Submission Request

The test cases of the “ArchiveSubmissionRequest” - function of the interface S.4 (sec. 4.5.4.1 Archive Submission Request) are also relevant here.

4.5.6.2 Archive Update Request

The test cases of the “ArchiveUpdateRequest” - function of the interface S.4 (sec. 4.5.4.2 Archive Update Request) are also relevant here.

4.5.6.3 Archive Evidence Request

The test cases of the “ArchiveEvidenceRequest” - function of the interface S.4 (sec. 4.5.4.5 Archive Evidence Request) are also relevant here.

4.6 Annex TR-ESOR-F

All requirements of Annex TR-ESOR-F are tested at the respective modules or interfaces.

4.7 Annex TR-ESOR-S

All requirements of Annex TR-ESOR-S are tested at the respective modules or interfaces.