Project Number:	**FP7-257123**
Project Title:	**CONVERGENCE**
Deliverable Type:	**Public**
Deliverable Number:	**D7.2**
Contractual Date of Delivery to the CEC:	**30.11.2011**
Actual Date of Delivery to the CEC:	**29.12.2011**
Title of Deliverable:	**Tools and Sample Applications Technical Specification**
Work-package contributing to the Deliverable:	**WP 7**
Nature of the Deliverable:	**Report**
Editors:	**Stelios Pantelopoulos and Panagiotis Gkonis**
Author(s):	**Stelios Pantelopoulos, Panagiotis Gkonis, (SIL), Alina Hang (LMU), Francis Lemaitre (FMSH), Angelo Difino (CEDEO), Daniel Sequeira (WIPRO), Silke Geisen, Thomas Hubner (MORPHO), Angelos – Christos Anadiotis, Aziz Mousas, Charalampos Patrikakis (ICCS), Mihai Tanase (UTI), Andrea de Polo, Sam Minelli (ALINARI), Andrea Detti (CNIT).**
Abstract:	**This deliverable defines technical specifications for the tools and sample applications of CONVERGENCE, describing also the APIs for the tools and the security framework for applications. APIs are then mapped to the functionality required by specific applications. We also describe the main steps in the planned implementation.**
Keyword List:	**Tools, applications, security, APIs, use cases, implementation**

Executive Summary

The goal of this deliverable is to define technical specifications for the tools and sample applications described in D.7.1. The deliverable begins by describing the APIs for the tools ([1]). It then goes on to describe the security framework for CONVERGENCE Applications (CoApps) outlining the functionality required to create a secure transaction environment (e.g. the use of smart cards, user identification, registration and authentication) and to protect the content (e.g. content encryption and decryption and signing).

Moving from specifications towards implementation, the next chapters of the deliverable are devoted to the four use scenarios that will be implemented in the trials. In each case, we map the APIs described in the previous chapter to the functionality required by specific applications and to application-specific security requirements and describe the main steps in the planned implementation. At the end of each use case chapter, we list the technology engines used by the application and the functionality to be supported at each stage.

ANNEX A provides samples of the application code. ANNEX B provides a complete description of the API for the CONVERGENCE security framework (COSEC).

This deliverable takes into account some comments of the first year review, especially those regarding the CoNet.

TABLE OF CONTENTS

LIST OF TABLES

Glossary

Term	Definition
Access Rights	Criteria defining who can access a VDI or its components under what conditions.
Advertise	Procedure used by a CoNet user to make a resource accessible to other CoNet users.
Application	Software, designed for a specific purpose that exploits the capabilities of the CONVERGENCE System.
Business Scenario	A scenario describing a way in which the CONVERGENCE System may be used by specific users in a specific context or, more narrowly, a scenario describing the products and services bought and sold, the actors concerned and, possibly, the associated flows of revenue in such a context.
CA	Central Authority
CCN	Content Centric Network
Cl_Auth_SC	Client Authentication with Smart Card (Challenge Response)
Cl_Auth_User_Pw	Client Authentication with Username and Password
Clean-slate architecture	The CONVERGENCE implementation of the Network Level, totally replacing existing IP functionality. See “Integration Architecture” and ”“Overlay Architecture” and “Parallel Architecture”.
CoApp	The CONVERGENCE Application Level.
CoApp Provider	A user providing Applications running on the CONVERGENCE Middleware Level (CoMid).
CoMid	The CONVERGENCE Middleware Level.
CoMid Provider	A user providing access to a single or an aggregation of CoMid services.
CoMid Resource	A virtual or physical object or service referenced by a VDI, e.g. media, Real World Objects, persons, internet services. It has the same meaning of “Resource” and it is used only to better specify the term “Resource” when there is a risk of a misunderstanding with the term “CoNet Resource”.
Community Dictionary	A CoMid Technology Engine that provides all the matching

Service (CDS)	concepts in a user's subscription, search request and publication.
CoNet Provider	A user providing access to CoNet services, i.e. the equivalent of an Internet Service Provider.
CoNet Resource	A resource of the CoNet that can be identified by means of a name; resources may be either Named-data or a Named service access point.
Content-based resource discovery	A user request for resources, either through a subscription or a search request to the CONVERGENCE system (from literature). See "subscription" and "search".
Content-based Subscription	A subscription based on a specification of user's preferences or interests, (rather than a specific event or topic). The subscription is based on the actual content, which is not classified according to some predefined external criterion (e.g., topic name), but according to the properties of the content itself. See "Subscription" and "Publish-subscribe model".
Content-centric	A network paradigm in which the network directly provides users with content, and is aware of the content it transports, (unlike networks that limit themselves to providing communication channels between hosts).
CONVERGENCE Applications level (CoApp)	The level of the CONVERGENCE architecture that establishes the interaction with CONVERGENCE users. The Applications Level interacts with the other CONVERGENCE levels on behalf of the user.
CONVERGENCE Computing Platform level (CoComp)	The Computing Platform level provides content-centric networking (CoNet), secure handling (CoSec) of resources within CONVERGENCE and computing resources of peers and nodes.
CONVERGENCE Core Ontology (CCO)	A semantic representation of the CoReST taxonomy. See "CONVERGENCE Resource Semantic Type (CoReST)"
CONVERGENCE Device	A combination of hardware and software or a software instance that allows a user to access Convergence functionalities
CONVERGENCE Engine	A collection of technologies assembled to deliver specific functionality and made available to Applications and to other Engines via an API
CONVERGENCE Middleware level (CoMid)	The level of the CONVERGENCE architecture that provides the means to handle VDIs and their components.
CONVERGENCE	The Content Centric component of the CONVERGENCE

Network (CoNet)	Computing Platform level. The CoNet provides access to named-resources on a public or private network infrastructure.
CONVERGENCE node	A CONVERGENCE device that implements CoNet functionality and/or CoSec functionality.
CONVERGENCE peer	A CONVERGENCE device that implements CoApp, CoMid, and CoComp (CoNet and CoSec) functionality.
CONVERGENCE Resource Semantic Type (CoReST)	A list of concepts or terms that makes it possible to categorize a resource, establishing a connection with the resource's semantic metadata.
CONVERGENCE Security element (CoSec)	A component of the CONVERGENCE Computing Platform level implementing basic security functionality such as storage of private keys, basic cryptography, etc.
CONVERGENCE System	A system consisting of a set of interconnected devices - peers and nodes - connected to each other built by using the technologies specified or adopted by the CONVERGENCE specification. See "Node" and "Peer".
Dec_Key_Unwrap	Key Unwrapping and Content Decryption
DIDL	Digital Item Description Language
Digital forgetting	A CONVERGENCE system functionality ensuring that VDIs do not remain accessible for indefinite periods of time, when this is not the intention of the user.
Digital Item (DI)	A structured digital object with a standard representation, identification and metadata. A DI consists of resource, resource and context related metadata, and structure. The structure is given by a Digital Item Declaration (DID) that links resource and metadata.
Domain ontology	An ontology, dedicated to a specific domain of knowledge or application, e.g. the W3C Time Ontology and the GeoNames ontology.
Elementary Service (ES)	The most basic service functionality offered by the CoMid.
Enc_Key_Wrap	Encryption and Key Wrapping
Entity	An object, e.g. VDIs, resources, devices, events, group, licenses/contracts, services and users, that an Elementary Service can act upon or with which it can interact.
Expiry date	The last date on which a VDI is accessible by a user of the CONVERGENCE System.

Fractal	A semantically defined virtual cluster of CONVERGENCE peers.
Group_Sig	Group Signature
ICN	Information Centric Network
Identifier	A unique signifier assigned to a VDI or components of a VDI.
Integration Architecture	An implementation of CoNet designed to integrate CoNet functionality in the IP protocol by means of a novel IPv4 option or by means of an IPv6 extension header, making IP content-aware. See “Clean-state Architecture”, “Overlay Architecture”, “Parallel Architecture”
IP	Identity Provider
License	A machine-readable expression of Operations that may be executed by a Principal.
Local named resource	A named-resource made available to CONVERGENCE users through a local device, permanently connected to the network. Users have two options to make named-resources available to other users: 1) store the resource in a device, with a permanent connection to the network; 2) use a hosting service. In the event she chooses the former option, the resource is referred to as a local named-resource.
Metadata	Data describing a resource, including but not limited to provenance, classification, expiry date etc.
MPEG eXtensible Middleware (MXM)	A standard Middleware specifying a set of Application Programming Interfaces (APIs) so that MXM Applications executing on an MXM Device can access the standard multimedia technologies contained in the Middleware as MXM Engines.
MPEG-M	An emerging ISO/IEC standard that includes the previous MXM standard.
Multi-homing	In the context of IP networks, the configuration of multiple network interfaces or IP addresses on a single computer.
Named resource	A CoNet resource that can be identified by means of a name. Named-resources may be either data (in the following referred to as “named-data”) or service-access-points (“named-service-access-points”).
Named service access point	A kind of named-resource, consisting of a service access point identified by a name. A named-service-access-point is a network endpoint identified by its name rather than by the Internet port numbering mechanism.

Named-data	A named-resource consisting of data.
Network Identifier (NID)	An identifier identifying a named resource in the CONVERGENCE Network. If the named resource is a VDI or an indentified VDI component, its NID may be derived from the Identifier (see “Identifier”).
Overlay architecture	An implementation of CoNet as an overlay over IP. See “Clean-state Architecture” and “Integration Architecture” and “Parallel Architecture”
Parallel architecture	An implementation of CoNet as a new networking layer that can be used in parallel to IP. See “Clean-state Architecture” and “Integration Architecture” and ”“Overlay Architecture”
PKI	Public Key Infrastructure
Policy routing	In the context of IP networks, a collection of tools for forwarding and routing data packets based on policies defined by network administrators.
Principal (CoNet)	The user who is granted the right to use a *CoNet Principal Identifier* for naming its named resources. For example, the principal could be the provider of a service, the publisher or the author of a book, the controller of a traffic lights infrastructure, or, in general, the publisher of a VDI. A Principal may have several Principal Identifiers in the CoNet.
Principal (Rights Expression Language)	The User to whom Permissions are Granted in a License.
Principal Identifier (CoNet)	The Principal identifier is a string that is used in the Network Identifiers (NID) of a CoNet resource, when the NID has the form: NID = <namespace ID, hash (Principal Identifier), hash (Label)> In this approach, hash (Principal Identifier) must be unique in the namespace ID, and Label is a string chosen by the principal in such a way that hash(Label) is unique for in the context of the Principal Identifier.
Publish	The act of informing an identified subset of users of the CONVERGENCE System that a VDI is available.
Publisher	A user of CONVERGENCE who performs the act of publishing.
Publish-subscribe model	CONVERGENCE uses a content-based approach for the publish-subscribe model, in which notifications about VDIs are delivered to a subscriber only if the metadata / content of those VDIs match

	constraints defined by the subscriber in his Subscription VDI.
Real World Object	A physical object that may be referenced by a VDI.
REL	Rights Expression Language
Resource	A virtual or physical object or service referenced by a VDI, e.g. media, Real World Objects, persons, internet services.
Scope (in the context of routing)	In the context of advertising and routing, the geographical or administrative domain on which a network function operates (e.g. a well defined section of the network - a campus, a shopping mall, an airport -, or to a subset of nodes that receives advertisements from a service provider).
Search	The act through which a user requests a list of VDIs meeting a set of search criteria (e.g. specific key value pairs in the metadata, key words, free text etc.).
Serv_Auth	Server Authentication without Smart Card
Service Level Agreement (SLA)	An agreement between a service provider and another user or another service provider of CONVERGENCE to provide the latter with a service whose quality matches parameters defined in the agreement.
Sig	Signature
Smart_Card Role_Auth_SC	Role Authentication towards Smart Card
SP	Service Provider
Subscribe	The act whereby a user requests notification every time another user publishes or updates a VDI that satisfies the subscription criteria defined by the former user (key value pairs in the metadata, free text, key words etc.).
Subscriber	A user of CONVERGENCE who performs the act of subscribing.
Timestamp	A machine-readable representation of a date and time.
Tool	Software providing a specific functionality that can be re-used in several applications.
Trials	Organized tests of the CONVERGENCE System in specific business scenarios.
Un-named-data	A data resource with no NID.
Us_Reg_IP	User Registration to Identity Provider
Us_Reg_SP	User Registration to Service Provider

User	Any person or legal entity in a Value-Chain connecting (and including) Creator and End-User possibly via other Users.
User (in OSI sense)	In a layered architecture, the term is used to identify an entity exploiting the service provided by a layer (e.g. CoNet user).
User ontology	An ontology created by CONVERGENCE users when publishing or subscribing to a VDI.
User Profile	A description of the attributes and credentials of a user of the CONVERGENCE System.
Versatile Digital Item (VDI)	A structured, hierarchically organized, digital object containing one or more resources and metadata, including a declaration of the parts that make up the VDI and the links between them.

1 Introduction

This deliverable provides technical specifications for the CONVERGENCE tools and sample applications originally described in D.7.1 ([1]) and analyzes the applications that will be used in the first phase of the CONVERGENCE trials. D.7.1 analyzed each application in terms of the functionality it required and of the engines used to implement this functionality. This deliverable goes one step further, specifying calls to Tools APIs and analyzing security constraints, see Figure *1-1* (Decomposition of CONVERGENCE Applications in CONVERGENCE Tools). The deliverable is divided into two main sections:

1. The first (chapters 2 and 3), describes the CONVERGENCE tools and the CoApps security framework, the APIs and additional functionality (e.g. license creation, insertion of metadata) they will use and their security requirements.
2. The second part (chapters 4,5,6 and 7), describes the four use case scenarios in terms of their security requirements, their requirements for user and registration rights, going on to specify the APIs that will be used for the creation of application VDIs.

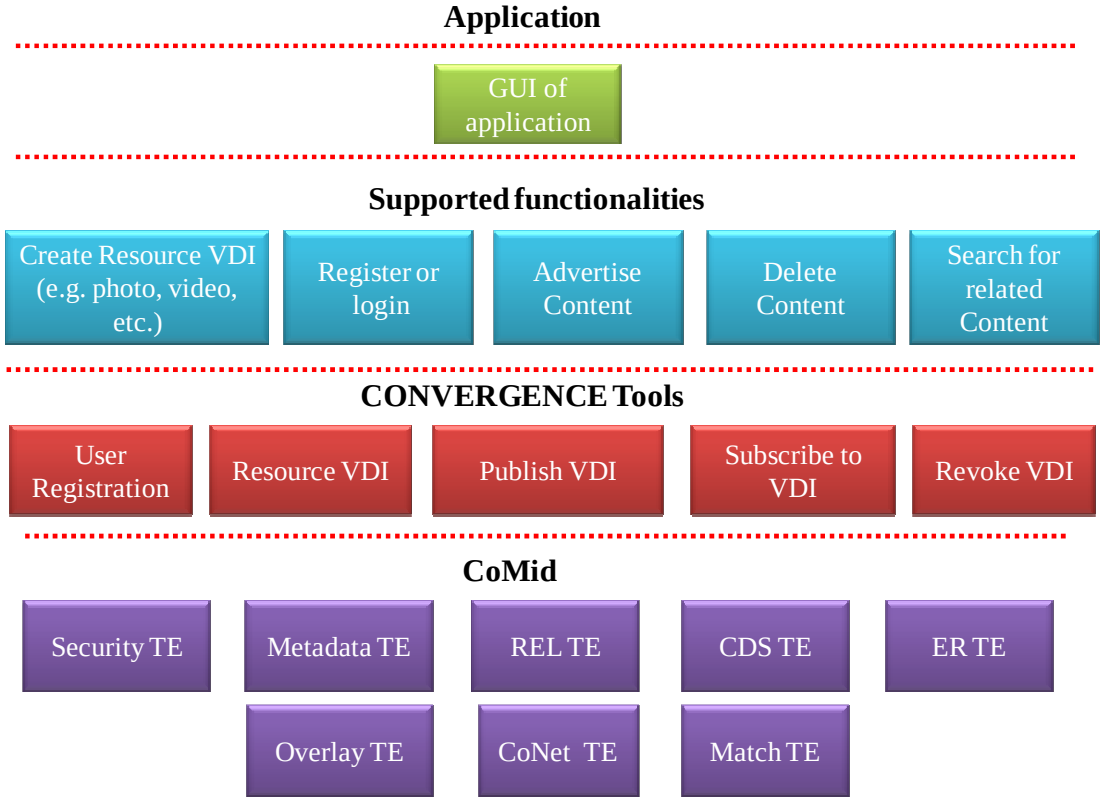

Figure 1-1: Decomposition of CoApps in CONVERGENCE Tools

The deliverable is organized as follows:

Chapter 2 describes the APIs for the basic tools as defined in D.7.1 and for the Resource Tool.

Chapter 3 specifies the CoApps security framework describing the use of smart cards, content encryption/decryption, key wrapping/unwrapping, user registration, client/server authentication and group signatures.

Chapters 4, 5, 6, 7 describe the four use case scenarios. Each description has the following structure

- Requirements:
  - User Registration and Rights: licensing requirements for different classes of user
  - Security: security constraints (e.g. anonymity, group signature, restrictions per user) for specific applications.
- Application VDIs:
  - Related APIs: Application specific APIs
  - Integration with the middleware: engines used to support specific application functionality

Chapter 8 summarizes some findings focusing on the way CONVERGENCE applications will exploit the CONVERGENCE concept of Information Centric Networking (ICN).

2 Description of the APIs for CONVERGENCE Tools

2.1 Introduction

We designed five basic CONVERGENCE Tools: User Registration, Resource VDI, Publish VDI, Subscribe to VDI, Revoke VDI. Other operations such as Creation of License, Metadata, or Event Report can be viewed as additional APIs to tools (namely Resource VDI, Publication VDI, Subscription VDI) since are common among them. For example, the APIs for the creation of license, ERR and metadata are common for Resource VDI, Publication VDI and Subscription to VDI Tools.

This chapter describes the APIs for the CONVERGENCE Tools listed in the following table.

Tools are categorized per different VDI type (P-VDI, S-VDI, R-VDI) (first column). In the second column, we report the list of tools as mentioned above: Creation and Storage of R-VDI (i.e. the Resource VDI Tool), Revocation (in all three VDIs), Creation and Injection of P-VDIs (i.e. the Publication VDI Tool), Creation and Injection of S-VDIs (i.e. the Subscription VDI Tool) and User Authentication and Identification (i.e. User Registration Tool).

Category	Tool	Engine
Resource VDI	Creation and Storage	Authenticate User
		Identify Content
		Media Framework (Encrypt Resource)
		Create License
		Describe Content
		Create ERR
		Create Content
		Sign Content
		Package Content
		Store Content
	Request	Authenticate User
		Request Content
		VDI Parsing
		Media Framework (Decrypt, Decode, Present Resource)
		Store Event
	Revocation	Authenticate User
		Send Revoke request
Publication VDI	Creation and Injection	Authenticate User
		Identify Content
		Media Framework (Encrypt Resource)

		Create Licence
		Describe Content
		Create ERR
		Create Content
		Sign Content
		Inject Content
		Store Content
	Revocation	Authenticate User
		Send Revoke request
Subscription VDI	Creation and Injection	Authenticate User
		Identify Content
		Media Framework (Encrypt Resource)
		Create Licence
		Describe Content
		Create ERR
		Create Content
		Sign Content
		Inject Content
		Store Content
	Revocation	Authenticate User
		Send Revoke request
User	Authentication	Authenticate User
	Identification	Identify User

Table 2-1: CONVERGENCE Tools and corresponding engines

2.2 User registration

This tool is responsible for the identification and storage of users and their credentials, enabling them to access and use secured functions inside CONVERGENCE. Users can register either with an Identity Provider (IP) or with a Service Provider (SP). Registering with an IP requires the user to provide his credentials in person. The process is assumed to be executed by a designated officer inside a secured and trustworthy area. The officer provides the user with a smart card containing his identity credentials and the IP's root certificate. Later the user will use an SP registration tool to register a pseudonym. The SP derives trust based on the IP user certificate. Whenever a user wants to access an application, he/she authenticates him/herself using this pseudonym. The SP's credentials are stored on user smart card during registration with the SP (this requires a user certificate issued by the IP).

2.2.1 Related APIs

2.2.1.1 Identity Provider

- **int identifyUser(String firstName, String lastName, Date birthday, String country, String locality, String organization, String pin)**. Stores the user on a database. Certificates and keys are generated for the user and his smartcard. The IP officer has to fill in the arguments. The function generates a UserID and returns it to the user.
- **void deleteUser(int userID)**. Completely deletes a user from the database, when the user is identified by a userID.
- **void deactivateUser(int userID).** Deactivates a user. The user is flagged as inactive.
- **void activateUser(int userID).** Activates a deactivated user.

2.2.1.2 Service Provider

- **int identifyUser(X509Certificate iPUserCertificate, String userId).** Stores the user on a database. Generates certificates and keys for the user and his/her smartcard. Provides the user's IP certificate as well as the pseudonym the user wants to register with. Generates and returns a userID for the user.
- **deleteUser(int userID).** Deletes a user from the data base.
- **deactivateUser(int userID).** Deactivates a user. Flags the user as inactive
- **activateUser(int userID).** Activates a deactivated user.

2.3 Content registration

This tool provides an identifier for a VDI or for a component of a VDI. The identifier allows the authentication of the VDI on later occasions. In line with industry practice, this is called Content registration. The names of the related engines (identification and authentication) are different.

2.3.1 Related APIs

- Identify Content
  - **String getIdentifier(DIDL vdi).** Returns the identifier of the VDI as a string. The identify content service provider keeps a record mapping the identifier to a hash value of the DIDL.
- Authenticate content
  - **Boolean AuthenticateContent(DIDL vdi).** Authenticates the VDI content.

2.4 Resource VDI

This tool allows a user to create and parse a Resource VDI (R-VDI) and provides methods to add and extract metadata, licenses, resource references and links to other VDIs. Additional methods allow users to package the VDI with resources, to sign and to store the VDI. These functionalities are supported by various middleware engines, the most important of which is the VDI TE. The metadata element of the R-VDI is created and parsed by the Metadata TE.

2.4.1 Related APIs

- Create R-VDI
  - **addKeyword (String keyword).** Adds a keyword description to the R-VDI, e.g. comedy.
  - **addFieldValue (String field, Object value).** Adds a field/value pair description to the R-VDI, e.g. genre, comedy. Note that value is of type object. This enables the use of non-String values, such as double or integer values and dates. This way the value preserves its data type.
  - **addTag (String tag).** This method adds a tag description to the R-VDI. Unlike keywords, tags are URIs backed up by a controlled vocabulary.
  - **addStructuredMetadata (StructuredMetadata structuredMetadata).** The method adds structured metadata descriptions to the R-VDI. Users may create structured metadata objects using the Metadata APIs. Each object consists of an XML description based on an XML schema e.g. MPEG-7 or RDF/XML descriptions based on ontologies.
  - **addLicense (License license).** Adds a license to the R-VDI. The user can create the license object with the License APIs, stating the issuer, grants to principals etc. Note that the R-VDI can embed multiple licenses.
  - **addSignature (Signature signature).** Adds a signature to the R-VDI. The user can create the signature object with the Security TE. Note that the signature is not a simple hash of the VDI, but a complex XML signature.
  - **addResource(String resourceUrl, String mimeType).** This method adds a resource reference descriptor to the RVDI. An optional mimeType parameter allows users to specify the content type.
  - **addRelationship(String relationshipType, String vdiId).** Adds a relationship descriptor stating a semantic relationship between another VDI and the VDI to be created. The relationshipType can be an URI identifying ontology relationship. However, this is not mandatory.
  - **setStartDate (Date startDate).** Sets the start date for the validity of the R-VDI.
  - **setExpiryDate (Date expiryDate).** Sets the expiration date for the R-VDI.
  - **DIDL generateRVDI ().** This method makes it possible to create an R-VDI, provided the metadata, licenses, etc. have already been inserted. The Resource VDI Tool

employs the Orchestrator TE to build a chain of engines, fuse the data submitted by the user and generate a valid VDI object.

- Parse R-VDI
  - **String getIdentifier().** Returns the identifier of the R-VDI.
  - **String getSequenceIdentifier().** Returns the sequence identifier of the R-VDI.
  - **List<String> getKeywordList().** Returns the keywords that describe the R-VDI.
  - **List<String> getTagList().** Returns the tags that describe the R-VDI.
  - **Map<String,List<Object>> getFieldValueMap().** Returns all field value descriptions in a map object. Users may add more than one value, hence the corresponding value list object.
  - **getStartDate().** Returns the start date for the validity of the R-VDI.
  - **getExpiryDate().** Returns the expiry date for the R-VDI.
  - **List<StructuredData> getStructuredDataList().** Returns a list containing all structured data descriptors for the R-VDI. The user can use the Metadata APIs to parse the metadata further.
  - **List<License> getLicenseList().** Returns a list containing all licenses embedded in the R-VDI. The user can use the License APIs to parse the licenses.
  - **List<Signature> getSignatureList().** Returns a list containing all signatures embedded in the R-VDI. The user can use the Security TE to parse the signatures.
  - **Map<String,String> getResourceList().** Returns a map object containing resource references and their MIME type.
  - **List<Relationship> getRelationships(DIDL rvdi).** Returns a list of the relationships of the VDI in a Relationship object which contains the relationship type and the relevant VDI.
- Store R-VDI
  - **Boolean store(String filePath).** Stores the VDI in the specified filePath location. If the user has not done so already, the tool generates the R-VDI.
- Advertise R-VDI
  - **Boolean advertise (String filePath).** Advertises an R-VDI when the R-VDI is located in a specific file path identified by the filePath spring. If there is no valid filepath string, the R-VDI is generated and stored using the store method.
- Package R-VDI
  - **Boolean pack (byte [][] resources).** Uses the MP21FF TE to packages a R-VDI with resources generating a MP21 file. The resources are passed as an array of byte[].Calls the generateRVDI method as well if the user has not yet created the R-VDI.

- **Byte [][] unpack(String filePath).** Unpacks any resources packed in the RVDI. The method returns an array of bytes [].

2.5 Publish VDI

This tool allows a user who has successfully created a VDI to create, inject and parse a Publication VDI (P-VDI), providing methods to add and parse metadata, licenses, and event report requests. Other methods allow users to inject a P-VDI into the overlay, to add a digital signature and to store it. These functionalities involve various middleware engines, the most important being the VDI TE. Metadata elements are handled by the Metadata TE. The Publish VDI tools shares several methods with the Resource VDI Tool. Most of these methods are supported by the same middleware engines in both tools.

2.5.1 Related APIs

- Create and Publish P-VDI
  - **addFractal(String fractal).** Adds a destination fractal to the P-VDI, identifying the fractal with a string (e.g. photo fractal). This is later used by the OverlayTE to propagate the P-VDI to the semantic overlay.
  - **addERR(ERR err).** Adds an event report request associated with the P-VDI. To create the ERR the user uses the Event Report Tool.
  - **DIDL generatePVDI().** Generates the P-VDI incorporating metadata, licenses, ERRs etc. previously passed by the user. During the generation process, the Orchestrator TE builds a chain of engines and fuses all the submitted data to form a valid VDI object.
  - **Boolean publish().** Publishes the VDI to the CONVERGENCE cloud.

The API also contains methods for addKeyword, addTag, addFieldValue, addStructuredMetadata, addLicense, addSignature, setStartDate, setExpiryDate. These methods are defined in the same way as in the Resource Tool.

- Parse P-VDI
  - **List<String> getFractalList(DIDL pvdi).** Returns a list of strings indicating the fractals that the P-VDI belongs to.
  - **List<ERR> getERRList(DIDL pvdi).** Returns a list of strings containing the Event Report Requests defined by the user when publishing the P-VDI.

The API also contains methods for getIdentifier, getSequenceIdentfier, getKeywordList, getTagListgetFieldValueMap, getStructuredMetadataList, getStartDate, getExpiryDate, getLicenseList, getSignatureList. These methods are defined in the same way as in the Resource Tool.

2.6 Revoke VDI

This tool allows CONVERGENCE users to revoke a P-VDI or a S-VDI from CoNet or CoMid. The tool uses CoNet TE for CoNet revocation and the OverlayTE to propagate the revocation request to peers.

2.6.1 Related APIs

- **Boolean revokeFromConet(String nid).** Revokes a VDI with nid from CoNet.
- **Boolean unpublishFromComid(String vdiId).** Revokes a VDI with vdiId from CoMid.

2.7 Subscription to VDI

This tool allows a user to create and parse a Subscription VDI (S-VDI). The tool provides methods for adding, and extracting conditions, licenses, event report requests. Additional methods allow the user to inject the S-VDI into the semantic overlay, to add a digital signature or to store it. These functionalities are supported by various middleware engines, the most important being the VDI TE. The creation and parsing of metadata is handled by the Metadata TE. Subscription queries are formulated using the MPQF TE.

2.7.1 Related APIs

- Create and Publish P-VDI
  - **addFractal(String fractal).** Adds a destination fractal to the S-VDI, identifying the fractal with a string (e.g. photo fractal) which the OverlayTE uses to propagate the S-VDI to the semantic overlay.
  - **addKeywordCondition(String keyword).** Adds a keyword condition to a user subscription query.
  - **addTagCondition(String tag).** Adds a tag condition to a user subscription query.
  - **addFieldValueCondition(String field, String operator, Object value).** Adds a field value condition to a user subscription query. The operator option allows users to express inequality conditions using the greaterThan or lessThan operators. Note that value is of type object. This makes it possible to preserve its datatype e.g. Date.
  - **addTripleCondition(String subject, String relationship, String operator, String Object).** The method allows the creation of a semantic subscription expressed as a triple pattern. The optional operator option allows users to express inequality conditions.
  - **addERR(ERR err).** Adds an event report request associated with the S-VDI. The user creates the ERR using the Event Report Tool.

- **DIDL generateSVDI().** Creates the S-VDI using metadata, licenses, errs etc. previously passed by the user. The Orchestrator TE then builds a chain of engines and fuses all the submitted data into a valid VDI object.

- **Boolean subscribe ().** Publishes the S-VDI to the CONVERGENCE cloud.

The API also contains methods for addLicense, addSignature, setStartDate, setExpiryDate are similar to the ones in the Resource VDI Tool. These methods are defined in the same way as in the Resource Tool.

- Store S-VDI

  - **Boolean store (String filePath).** Stores a S-VDI in a specific filepath.

- Parse S-VDI

  - **List<String> getFractalList(DIDL svdi).** Returns a list of strings indicating the fractals that the S-VDI belongs to.

  - **List<String> getKeywordConditionList ().** Returns a list with the keyword conditions for the subscription.

  - **List<String> getTagConditionList ().** Returns a list with the tag conditions for the subscription.

  - **List<FieldValueCondition> getFieldValueConditionList ().** Returns a list with the field value conditions for the subscription. The FieldValueConditon object contains the field, the value and the comparator operator.

  - **List<TriplePatternCondition> getTripleConditionList().** Returns a list with the triple conditions for the subscription. The TriplePatternConditon object contains the subject, relationship, object and the comparator operator.

  - **List<ERR> getERRList(DIDL svdi).** Returns a list of strings with the event reports defined by the user when publishing the S-VDI.

The API also provides methods for getIdentifier, getStartDate, getExpiryDate, getLicenseList and getSignatureList. These methods are defined in the same way as in the Resource Tool.

2.8 Browse Event Report

This API allows a CONVERGENCE user to browse a previously created event report

- Browse ERR

  - **List<ER> getERRList**(ERR err). Returns a list of event reports corresponding to the ERR given as a parameter

2.9 Event Report Creation

These APIs allow a user to create and parse an ER (Event Report) and/or an ERR (Event Report Request). The user begins by creating the ERR and then associates the ERR with a Resource, Subscription or Publication VDI.

2.9.1 Related APIs

- Create ERR
  - **ERR createERR(EventTriggerType verb).** The user provides the verb as the event trigger type. The user begins by creating the ERR and then associates the ERR with the Resource, Subscription or Publication VDI. The ERR contains the peer's service access point name.
- Parse ERR
  - **String getERRIdentifier(ERR eventReportRequest).** Parses an ERR and returns the ERR identifier.
  - **String getVerb(ERR eventReportRequest).** Parses an ER and returns the verb (match, notify etc.).
- Parse ER
  - **String getERRIdentifier(ER eventReport).** Parses an ER and returns the ERR identifier.
  - **String getPublicationVDIId(ER eventReport).** Parses an ER and returns the publication VDI identifier.

2.10 License Creation

These APIs allow users to create and parse licenses associated with the resources of specific VDIs. Licenses are expressed in the Rights Expression Language (REL) and are serialized with the help of the REL TE. Additional methods support the verification of a license's validity and checking of the rights of a principal on a resource.

2.10.1 Related APIs

- Create License
  - **KeyHolder createKeyHolder(KeyInfoType keyInfoType).** Creates a new key holder every time a user wishes to define a principal for a resource. A KeyHolder principal is identified by a KeyInfoType, which may include certificate data.
  - **PropertyPossessor createPropertyPossessor (String property).** Creates a principal of type PropertyPossessor. Unlike KeyHolder principals, PropertyPossessor principals do not have a certificate but are identified by a particular property. This may be either a URI or an attribute.

- **Condition createCondition(int conditionType, Map parameters).** Creates conditions, including territory and temporal conditions. The parameters for each type of condition are passed in the form of a parameter value map.

- **setIssuer(KeyHolder keyHolder).** Sets the issuer of a license.

- **addGrant(KeyHolder keyHolder, PropertyPossessor propertyPossessor, Right right, Map<Integer,String> resourceType2URI, Condition condition).** Adds a new Grant in the license. Each Grant refers to a Principal, a property possessor or a keyholder, granting him/her a Right on the specified resource under specified conditions. The list of Rights that a user may specify in the license are specified in the definition of the REL and include rights to adapt, copy or play a resource.

- **setSignature(SignatureType signatureType).** Sets a signature for the license.

- **License generateLicense ().** Generates a new license that contains all submitted data.

- Parse License

  - **Issuer getIssuer().** Returns the issuer of a license.

  - **List<Grant> getGrantList ().** Returns a list with containing all the grants specified in the license.

  - **SignatureType getSignature().** Returns the signature of a license. Users can check the signature by calling the SecurityTE.

- Check License

  - **Boolean verifyLicense ().** Verifies the validity of the license. The tool users the REL TE to process the license, to check its integrity and to check the validity of REL statements contained in the license.

  - **AuthorizationRepsonse checkLicense (Principal principal, Right right, String resourceRef).** Processes the license and checks the privileges of a principal on a certain resource. The user provides a principal, a right and a resourceRef. Then the tool uses the AuthorizationManager in the REL TE to check the rights granted to the principal, including geographical and temporal conditions. Finally, it returns an AuthorizationResponse to the user.

2.11 Metadata

These APIs allow users to create and parse complex descriptions of XML metadata. Descriptions may be based either on XML schemata, like MPEG-7, or on ontologies and the Resource Description Framework. In this case the output consists of RDF/XML. Ontology entities are provided by the CDS TE, making it easier for users to build rich semantic descriptions.

2.11.1 Related APIs

- Request Metadata

- **List<StructuredMetadata> requestMetadata(String request, String type, String modelURI).** Returns a list of metadata entities that satisfy the parameters passed by the user. The modelURI parameter specifies the model where the entity will be searched(e.g “http://purl.org/dc/terms/” for Dublin Core terms). The request parameter specifies the prefix of the entity label. The type parameter is used to support cases in which the model is an ontology, and it is necessary to specify the type of a particular entity.

- Create Metadata

  - **addRDFTriple (String subject, String property, Object object).** Adds a RDF triple to the description. The user passes a subject, a relationship and an object and the tool transforms it to a RDF statement. If the subject is null, the result is a blank node. Note that the object is of anyType, allowing users to submit also non-String values to datatype properties.

  - **addXMLStatement (String field, Object value).** Adds a field value statement to the description. User passes a field URI (e.g. http://purl.org/dc/terms/creator) and a value object of anyType.

  - **StructuredData generateStructuredData ().** The method returns a structuredMetadata object containing the full description submitted by the user.

- Parse Metadata

  - **List<RDFTriple> getRDFTriples().** Returns the RDF triples contained in the description.

  - **Map<String, Object> getFieldValueMap().** Returns a map containing all the XML statements in the description. The keys of fields are qualified names of XML elements.

3 Applications Security Framework

From the description of functional requirements for the four use cases scenarios in D.7.1, ([1]) it is clear that operations described in the scenarios require enforcement of security. Examples include user registration and identification, the creation of secure sessions for data exchange, content encryption, as well as the creation of group signatures and the creation of "trusted" groups of users. In the following sections we briefly describe the main security functionality we intend to provide. This will include:

- Use of smart cards
- Creation of "trusted" groups through a PKI
- Content encryption and key wrapping to secure a VDI or other content
- Content decryption and key unwrapping to decrypt encrypted VDI or other content
- User registration with Identity and Service Providers
- Client and server authentication with and without a smart card
- Signing with a Group Signature

Related APIs (in bold letters in this chapter) are described in ANNEX B.

3.1 Use of smart cards in Convergence

Cryptographic software, or at least its essential components, should always run in a trustworthy environment. Naturally, the easiest assumption is to regard the entire network as safe, and to assume that registered devices (laptops, PCs, etc.), once checked, will remain secure devices and can be trusted. Unfortunately, the opposite is true. A network is anything but secure, and devices, even if initially secure, can be tampered with by fraudulent users, or manipulated by third parties. Many of these problems can be solved, or at least mitigated, by the use of smart cards as secure hardware modules. Smart card software is almost completely safe against tampering. This means that an attacker will find it very hard to access confidential content stored or processed on a well-designed system. In this way, the smart card serves as a reliable outpost for the service provider, which continues to control it, even though it is permanently in the hands of an end-user.

An additional advantage of smart cards is that they can store secret information which is not available anywhere else, not even on the manufacturer's or card issuer's site. A typical example is a signature key: such keys are generated on-card, are unknown to any outside entity, and always remain in the physical possession of the card holder. The card holder can therefore be certain that no one - not even the service provider or the card manufacturer - can forge her signature without physically accessing her smart card.

In CONVERGENCE, the user gets his personal smart card during registration (see section 3.5).

The smart card stores sensitive information such as:

- User identifier
- Secret key
- An automatically generated PIN
- A signature key
- A group signature key

Additionally, the certificate given to the user after registration to IP (User registration) is stored on his smart card as well as by the IP who handles its subsequent distribution to key servers.

Each user's smart card contains at least three types of (conventional) asymmetric key pairs:

1. Key pair for user authentication
2. Key pair for signing (VDIs, content, etc.)
3. Key pair for decryption (hybrid encryption/decryption, i.e. typically "un-encapsulating" symmetric keys with which content has been encrypted)

Users may generate additional key pairs when registering at Service Providers for specific applications. In each case:

- Encryption and decryption are based on asymmetric cryptography schemes meeting the most recent security requirements prescribed in official regulations (e.g. RSA with SHA-256 and bit length 2048).
- The user's signature key is generated on the card, and certified during registration (this is achieved without revealing the corresponding secret key)
- The user's private key never leaves the smart card, and is not known to any entity outside the smart card (not even to the user himself)
- The certificate containing the user's public key is kept for subsequent distribution, and stored on the user's smart card.

User authentication in short (more details are in section 3.6)

User provides:

- Personal smart card
- PIN

User needs a smart card reader usable on any OS (Windows, Mac)

User authenticates himself/herself to the network

If the authentication is successful, the application creates a session which

- is encrypted with session keys that are derived from key material from the authentication procedure
- stays alive during processing of the VDI until it is signed

3.2 "Trusted groups" through PKI

Certificates are a standard means to inherit trust in a public key infrastructure (PKI). Certificates have two basic functions:

1. To bind a public key to an entity (human user, device, network node, institution, etc.) in a trusted way.
2. To allow a trustworthy authority to associate attributes and entities.

Certificates are issued by Certificate Authorities (CAs) whose defining property is that users trust them. Usually an entity acts as a CA towards a specific group of users specified in a well-defined policy. In this way, a government organization can act as a CA for the citizens of a particular country, or a manufacturer can take on this role for a particular group of retailers and customers.

The architecture of the CONVERGENCE provides for three levels of certificates:

1. (Self-certified) Root certificates for each CONVERGENCE Identity Provider.
2. CONVERGENCE Service Provider and Application certificates issued by an Identity Provider.

3. End-User certificates issued by Identity Providers, or Service Providers.

When a manager wishes to create a "trusted group" he/she can become the CA for the group. Certificates are issued and managed using functionalities provided in the CoSEC-API (see ANNEX B Brief description of the CoSEC API).

3.3 Content Encryption and Key wrapping

This and the following section describe the workflow for "hybrid" content encryption and decryption. In the case of encryption, a (potentially large) content file is encrypted with a randomly generated "one-time" symmetric encryption key. [Note: Each new content file is encrypted with a new encryption key.] The key itself is then encrypted with the public key of the recipient ("wrapped") and embedded in a license. The length of the key depends on the chosen RSA configuration. The minimum – guaranteed even with RSA-1024 - is 496-bit.

Server	Recipient
symmKey = **generateRandomKey** (*algorithm*, *keyLength*)	
encrypt (*content*, *symmKey*, *algorithm*)	
Upload encrypted content	→
Obtain recipient's certificate	←
wrappedsymmKey = **wrap** (*symmKey*, *algorithm*, *keyAlias*, *certificate*), encrypt *symmKey* with the public key associated to *keyAlias* extracted from the certificate	
Embed *wrappedsymmKey* in license	
Send license to recipient	→

Table 3-1: Content encryption work flow

3.4 Key unwrapping and content decryption

A (potentially large) content file has been encrypted with a randomly generated "one-time" symmetric encryption key, embedded in a license as described in the previous section. Note that the wrapped key is actually an "encrypted token" and will expand to "full RSA-length" (e.g. 1024 or 2048 bit) even if the actual key is only 16 bytes long. The wrapped key is extracted from the license and subsequently transmitted to the client's smart card for "unwrapping" (i.e. decryption using the recipient's private key stored on the smart card). The private key itself never leaves the smart card. Unwrapping is authorized by the user's PIN. Using the smart card for unwrapping makes it possible to implement further constraints before unwrapping. For instance, unwrapping might require on-card certificate validation before the unwrapped key is delivered to the client PC. For simplicity of description, the workflow below does not include this option.

	Client		Smart Card
→	Obtain license, extract wrapped key *wrappedsymmKey*		
	Ask user for PIN		
	unwrap (*wrappedsymmKey*, *algorithm*, *keyAlias*, *PIN*)	→ ←	decrypt *wrappedsymmKey*, return *symmKey*
→	Download encrypted content *encryptedContent*		
	decrypt (*encryptedContent*, *symmKey*, *algorithm*)		

Table 3-2: Content decryption work flow

3.5 User Registration

The first step in every scenario is to register the user and his/her credentials with an Identity Provider. Once registered, the user receives his/her certificate and a personal smart card. The smart card stores the key pair for the users and the certificate which the user will use when he/she needs to authenticate himself/herself.. He/she will also receive a unique UserID.

Steps	Client	Identity Provider
1.	The user accesses the trusted Identity Provider (IP) and registers himself/herself (usually in person). For this he has to show his/her correct/ true credentials (e.g. identity card).	
2.		The Identity Provider (IP) gives the user a smart card, which stores a key pair for the user and the associated certificate. In addition the certificate itself includes the public key of the user, personal data (e.g. name) and a unique identifier which the IP has assigned to the user. (Length: 16 Byte). The Certificate is given to the user, stored on his/her smart card, and stored at the Identity Provider who will manage its subsequent distribution to key servers.

Table 3-3: RegisterUser to IdentityProvider

After registering with the identity provider, the user can use his/her smart card to register himself/herself with a service provider, using the protocol described below.

Steps	Client	Service Provider
1.	The user wants to register to an Application managed by a particular Service Provider (SP). He/she sends a	

Steps	Client	Service Provider
	command to the SP specifying his/her username (e.g. a pseudonym), and certificate, and the name of the application he/she wishes to register for.	
2.		The SP starts a client authentication process, verifying the user's Certificate and performing a challenge-response protocol to verify the identity of the user.
2b	(Optional: server authentication)	

---------- A secure session is established ----------

3.		The SP checks if the favoured username is free or assigned to somebody else. If it is assigned to somebody else, the server asks for a new favoured username. If it is free, the service provider requests the public key of the client.
4.	The client generates a private/public key pair and sends the public key to the service provider	
5.		The SP generates a random challenge and sends it to the client.
6.	The client receives the challenge (receiveChallenge), signs it with the private key, and returns the signed challenge as a challengeResponse	
7.		The SP verifies the challengeResponse (verifyChallenge). If it fails, the protocol is aborted. If it passes (i.e. the SP is convinced that the client hold the private key corresponding to the public key sent previously), the SP generates a certificate for the favoured username (pseudonym), and signs it. The certificate includes the pseudonym, the client's public key, the application and the name of the service provider. The certificate is sent back to the client, and stored at the SP.

Table 3-4: Register User to Service Provider for specific applications

3.6 Authentication

After the registration to the IP and SP, the user is able to authenticate him/herself. The CONVERGENCE demo architecture includes two options for authentication, described in the sections below.

3.6.1 User Name and Password

The user specifies his /her desired username and password during registration with the service provider. Later he/she can use the same username and password for authentication.

1		The client wants to take an action or to login.
2	The server generates an AuthenticateUserRequest and sends it to the user	
3		The user responds (ClientResponse) with his/her login and password
4	The server returns a message stating whether the authentication was a success or a failure	
5		The user receives a success or failure message

Table 3-5: User Name and Password Authentication

3.6.2 Authentication via smart card

The client PC is operated by a user trying to access a service. The server seeks to authenticate the user before authorizing access. Authentication is based on the user's digital signature. The user's smart card, connected to the client PC acts as a "crypto box" providing a secure repository for the user's private signature key. See below.

Server		Client		Smart Card
Send AuthenticateUser Request	→			
		getCertificate(*keyIdentifier*)	→ ←	Retrieve User Certificate
verifyCertificate(*certificate*), extract client's public key	←	Send User Certificate		
getChallenge(), send *serverChallenge* to client	→	receive *serverChallenge*		
		ask user for PIN		
		clientAuth(*serverChallenge*, *keyIdentifier*, *PIN*)	→ ←	compute authentication token *cardAuthToken*
clientTokenVerify(*cardAuthToken*, *keyAlias*)	←	send *cardAuthToken* to server		
Send AuthenticateUser Response	→			

Table 3-6: User Name and Password Authentication

3.6.3 Server Authentication (without Smart Card)

This section describes the protocol for a server authenticating itself to a client using a digital signature. The client PC is operated by a user who wishes to use an authentication procedure to ensure that the server he/she is addressing is genuine.

Server		Client
	←	Send AuthenticateServer Request
Send server certificate (containing server's public key)	→	
		verifyCertificate(*certificate*) extract server's public key
		generate random challenge
receive challenge	←	Send challenge to server
serverAuth(*Challenge*, *keyAlias*, *password*), obtain authentication token *serverAuthToken*, send token to client	→	
		serverTokenVerify(*serverAuthToken*, *keyAlias*) verify *serverAuthToken*
	←	Send AuthenticateServer Response

Table 3-7: Server Authentication (without Smart Card)

3.6.4 Server Authentication (with Smart Card Role Authentication)

This section describes a variant of the protocol described in the previous section but this time with "Role Authentication" towards the user's smart card. This protocol is used for applications where an external entity (here: the server) requests data/services from the smart card. (e.g. when the server wants to read confidential personal data from the smart card). The server is thus associated with a specific "role". Usually this means granting the smart card specific access rights, encoded in the server's CV-certificate. In addition, for such role authentication, the server needs to authenticate itself towards the smart card itself and not just the client PC. This means that it is the smart card and not the client PC that generates the random challenge. It also means that the smart card has to verify the certificate to check the access rights granted to the requesting server.

Server		Client		Smart Card
	←	Send AuthenticateServer Request		
Send CV-certificate containing server's public key	→			
		verifyCVCertificate(*CVcertificate*)	→	verify the CV-certificate, extract server's public key into card RAM
		getCardChallenge()	→ ←	generate random challenge
receive card challenge	←	Send card challenge to server		
serverAuth(*cardChallenge*, *keyAlias*, *password*), obtain authentication token *serverAuthToken*, send token to client				
		serverTokenVerify(*serverAuthToken*, *keyAlias*)	→ ←	verify *serverAuthToken* on-card
	⇄	Send AuthenticateServer Response		

Table 3-8: Server Authentication (With Smart Card)

3.7 Signing

3.7.1 Signature

Digital signatures are supported by an interface providing a broad range of functionality including generation and verification of hash-values, issuing of digital certificates and verification of certificates. The table below illustrates the protocol for the creation of a digital signature.

Client	Smart Card
The Client wants to digitally sign a message	
	The Smart Card receives a (partial) hash of the message and generates a digital signature with **generateCardSignature**.
The Client receives the signed messaged, which he sends to a certain person (or server)	

Table 3-9: Signature workflow

The receiver of the message verifies the message with **verifySignature** (see ANNEX B Brief description of the CoSEC API).

3.7.2 Group Signature (GS) Schemes in a nutshell

A group signature is an anonymous digital signature issued on behalf of a group. Each group member is endowed with his/her own private key. However, there is only one public key for the entire group. A group master (operating in the spirit of a CA) sets up the group and holds a master key, which he/she keeps absolutely secret. Group signatures are in some way double-faced, providing two apparently contradictory features.

- Anonymity. A verifier can verify whether a group signature has been issued by a legitimate group member. However the actual identity of the individual group member responsible for a given signature is hidden both from the verifier and from other group members. Different group signatures cannot be linked, i.e. it is impossible to decide whether two group signatures have been generated by the same individual.
- Traceability. Under well-specified conditions, the Group Master can use the master key to unveil the individual responsible for a given signature.

Individual group members may be revoked without breaking up the remaining group. It is also possible to add new members to a group. The group signature scheme chosen for CONVERGENCE allows revocation of group members without updating the private keys of the remaining members. In cases where smart cards are used, this avoids the cumbersome process of updating many cards in the field. The disadvantage of the scheme is that it makes it necessary to maintain a "black list" of (anonymous) revocation tokens.

The API for group signatures considers three categories of "users":

1. The group master – who administers the group and holds the master key
2. Group members – members of the group who sign anonymously on behalf of the entire group
3. Outsiders – who may verify group signatures.

3.7.2.1 Signing a message anonymously using a smart card

This section describes the workflow for a group member anonymously signing a message on behalf of an existing group. In this workflow we assume that the group member has a smart card containing the public parameters for the group and the card holder's private group signature key. In this example, the signer will not notice any difference with respect to the normal procedure for digital signatures (except the process may take a little more time). Given that signature operations are performed on card, group signatures are secure, even when the user attaches the card to an untrustworthy terminal.

	Signer (group member)		**Smart Card**
	Prepare message to be signed		
	Ask member for PIN, authenticate to smart card	↔ →	
	sign_message_GS_SC (*message*, *GSAlias*, *PIN*)		identify group-parameters by *GSAlias*, access *SKmember*, compute *signature*, return *signature*
←	issue signature	←	

Table 3-10: Workflow for anonymous signing

3.7.2.2 Group Signature Verification

This section describes the workflow for group signature verification. The scheme consists of two steps:

1. Signature check (cryptographic validity)
2. Revocation check

The revocation check comes in two flavours:

- *Local revocation-check*: If a revocation list is available to the verifier, he or she can use it directly[1]
- *Remote revocation-check*: If no revocation list is available, the verifier needs to send the signature to a trusted server to be checked.

The table below describes a verification workflow with local revocation-check. The verifier has obtained a tuple consisting of a group *signature* and a *message* the signature claims to be valid on.

(Outside) verifier	Signature Issuer
obtain_parameters_GS (*GSAlias*) store *GSpublicparameters* locally addressed by *GSAlias*	←
perform_signaturecheck_GS (*message, signature, GSAlias*) reject if "false" is returned	
obtain the current revocation list for the group *GSAlias*	←
perform_revocationcheck_GS (*signature , RevocationList, GSAlias*) reject if "false" is returned	
Accept the signature only if both checks have been passed	

Table 3-11: Verification workflow with local revocation-check

[1] In some cases, there may be good reasons not to maintain a publicly available list of revocation tokens. One of these is that such a list would make it possible to link all the signatures of the revoked group measure including signatures issued prior to the revocation. In some applications, this would not be desirable.

4 Specifications and Technical Implementation of "Photos in the Cloud and Analyses on the Earth" (real world trial 1)

4.1 Brief description of use case

The goal of this use case is to test whether CONVERGENCE can provide useful support for new business models in the photography business. The goal is to make it easier for photographers to contribute and describe photos, improve access for users and generally facilitate the management of relevant services. All photographs are represented by VDIs, each containing:

1. The photo itself (or a link to the photo) possibly accompanied by a low-resolution version of the photo
2. Metadata describing the photo, including the date, time and place where the photo was taken, legal data on the author and owner of the photo, technical data about the photo (camera, lens, shutter time, aperture, ISO etc.), historical data about the site represented in the photo, and other metadata contributed by Alinari staff and third parties
3. Licensing information representing the conditions at which photos can be licensed by a photographer to Alinari and by Alinari to an end user. Licensing information is represented using the CONVERGENCE REL.

Alinari manages the server used in the trial and the photo archive using a dedicated server that runs custom applications on top of the CONVERGENCE framework and the CONVERGENCE network. The dedicated applications provide Alinari staff with a user interface which makes it easy for them to create, publish, un-publish, describe and update the photos and VDIs. Free-lance photographers can access the Alinari service to create Resource VDIs and Publication VDIs. End-users can subscribe to and possibly buy photos using a local application connected to the Alinari server. The CONVERGENCE framework automatically prevents access to photos that have expired and performs garbage collection to purge expired copies from network storage.

4.2 Bird's eye view of the deployment framework

The "Alinari scenario" involves:

1. Three types of users:
    1. Photographer
        i. Freelance, independent of Alinari
        ii. With a business relationship (e.g. registered) with Alinari
    2. Alinari photo archive manager
    3. Customer of Alinari photo licensing service
2. The following devices:
    1. Alinari server containing copies of current Alinari photos and metadata in CONVERGENCE format (in this deliverable called Alinari* server)
    2. Cloud of CONVERGENCE peers in the overlay
    3. Various clients (peers) used by participants

The following data:

1. Photos from the Alinari server and Photo VDIs created from the Alinari photo metadata
2. Photos and Photo VDIs created by photographers

3. The following applications
    1. User registration
    2. Photographer publishes Photo VDI
    3. Rights holder un-publishes Photo VDI
    4. Alinari subscribes and downloads photo
    5. Photographer uploads Photo VDI to Alinari server
    6. Alinari publishes new Photo VDI
    7. Customer purchases interesting photos
    8. Customer visits Alinari Museum

4.3 Application Requirements

4.3.1 User registration and rights

4.3.1.1 User registration

There are two ways for a user to register to the Alinari application.

- Via smart card, providing the following information:
    - First name and last name
    - Valid email account
    - User Information:
        - ✓ Tel number (optional)
        - ✓ Agency name (optional)
            - Passport number
            - Signature
- Via web interface, providing the following information:
    - First name and last name
    - Valid email account
    - Password
    - User Information:
        - ✓ Tel number (optional)
        - ✓ Agency name (optional)

4.3.1.2 User rights

The table below, extracted from D.4.2 [3] describes the rights assigned to different classes of user.

#	Issuer	Principal	Type of VDI	REL verbs	*Comments*
1	Freelancer photographer	Alinari	R-VDI of Photo	GovernedCopy, GovernedAdapt, Post, License	*The photographer gives Alinari personnel the right to store the photo in their web-site and make adaptations*
2	Alinari personnel	Anybody	R-VDI of Photo	Play, Print	*A photo is available to the public for reprint*
3	Freelancer photographer	Fractal	P-VDI of Photo	Match, Notify	*The photo is advertised in CoMid*
4	Alinari personnel	Fractal	P-VDI of Photo	Match, Notify	*The photo is advertised in CoMid*

Table 4-1: User registration and rights for the Alinari scenario

4.3.2 Security Requirements

The first phase of the trials will test authentication with username/password. The later phases will also support the use smart cards. The following security requirements will apply.

1. Only registered users to the Alinari may upload photos to the Alinari server.
2. Alinari personnel must belong to a group certified by the Alinari Manager.
3. Freelancer photographers with a business relationship with Alinari must belong to a group certified by the Alinari Manager.
4. Users should be able to authenticate the Alinari server prior to a transaction, in order to be sure that they upload their photo to the correct server.
5. The Alinari server should be in position to authenticate a remote client belonging to an Alinari certified user, reducing the risk that users will upload improper material to the server.
6. Freelancer photographers with a business relationship with Alinari have to be able to sell an encrypted version of the photo; on demand, the Alinari Server has to be able to provide a license to decipher the photo.

4.3.3 Photo encryption/decryption

The Alinari server encrypts photos using hybrid encryption. A photo resource is first encrypted by a (fast) symmetric cipher using a fresh random symmetric key. The key is then "wrapped" (encapsulated) by encrypting it with the recipients' public key and embedded in a license, as described in Content Encryption and Key wrapping. The decryption key used for "unwrapping" is embedded in the smart cards and can only be released by authorized users (who are in physical possession of the smart card and authorize themselves with a PIN or

biometric feature). Photo encryption is performed in the Alinari Server whenever a photo is sold while photo decryption is performed on a local device.

The table below matches the security requirements described in paragraph Security Requirements to the security functionalities defined in Applications Security Framework (abbreviations explained in Glossary section).

Scenario requirement	Enc_Key_Wrap	Dec_Key_Unwrap	Us_Reg_IP	Us_Reg_SP	Cl_Auth_Us_Pw	Cl_Auth_SC	Serv_Auth	Smart_Card_Role_Auth	Sig	Group_Sig	PKI	REL
1			X	X	X	X	X	X				
2			X	X							X	X
3			X	X							X	X
4					X	X	X	X				
5					X	X						
6	X	X										X

Table 4-2: Security requirements for the Alinari scenario

4.4 Creation of application VDIs

This section describes the creation of the VDIs required by the application, with the help of the CONVERGENCE Tools described in Chapter 2. The discussion is limited to the applications to be deployed in the first phase of the CONVERGENCE trials. The corresponding implementation code can be found in ANNEX A.

4.4.1 Photographer uploads Photo VDI to Alinari server

In this first application, a freelance photographer wants to create a new photo VDI. Each such VDI will contain metadata related either to the photo (such as tags, date of capture, info related to the analysis of the image, etc.) or to the photographer. Additionally, the photographer will have to set rights over this digital resource (according to Table 4-1: User registration and rights for the Alinari scenario) with the help of the License Tool. After successful registration, he launches the Create Photo VDI application. The procedure for creating a new R-VDI, is described in the following steps:

1. *Initialize a new Resource VDI instance*
2. *Use the environment to fill in the appropriate metadata (tags)*
3. *Add an image reference to the resource VDI*

4. *Set an expiry date for the VDI*
5. *Store the VDI*
6. *Advertise the VDI*

4.4.2 Photographer publishes Photo VDI

In this second application, a photographer publishes a VDI. This involves the following steps:

1. *Initialize a new instance of the Publication VDI Tool, based on the R-VDI identifier*
2. *Set an expiry date for the VDI*
3. *Add the VDI to a specified fractal*
4. *Publish the VDI in the cloud*

4.4.3 Alinari Photo Archive Manager subscribes to and downloads photos

In the third application, the goal is to allow users to search for specific content by injecting S-VDIs into the cloud. The steps are the following:

1. *Initialize a new instance of the Subscription VDI Tool*
2. *Set a fractal for the S-VDI*
3. *Set an expiry date to the S-VDI*
4. *Add the keywords and search tags specified by the user*
5. *Add an event report (an event is triggered on every match)*
6. *Publish the S-VDI*

4.4.4 Rights holder un-publishes Photo VDI

In the fourth application, a rights holder wishes to un-publish a VDI from CoMid. He, therefore, sends an un-publish VDI message to the appropriate peers in the cloud. This removes the data associated with the specific VDI and send a confirmation to the rights holder.

4.5 Integration with the middleware

4.5.1 Engines per application

The table below (Table 4-3: Engines per application) summarizes the list of TEs required per each application in the first phases of the trial. Table 4-4: Supported functionalities per trial shows the functionalities that will be supported in the different phases.

Technology Engine	Photographer creates and publishes photo VDI	Alinari subscribes and downloads photo	Rights holder un-publishes photo VDI	Photographer uploads photo VDI
Metadata	X	X		
REL	X	X		X
ER	X	X		X
Security	X	X	X	X
Overlay	X	X		

CoNet		X	X	
CDS	X	X		X

Table 4-3: Engines per application

Functionality	Phase 1	Phase 2	Phase 3
Username/ Password User Authentication	X	X	
Smart card supported User Authentication			X
Single User Identification	X	X	X
Group identity creation by Alinari Manager		X	X
Manually generated metadata for the creation of Photo VDIs	X	X	X
CDS-support generated metadata for the creation of Photo VDIs		X	X

Table 4-4: Supported functionalities per trial

5 Specifications and Technical Implementation of "Videos in the Cloud and Analyses on the Earth" (real world trial 2)

5.1 Brief description of use case

The "Videos on the Cloud and Analyses on the Earth" scenario has been designed to improve the management of audiovisual archives and to exploit the potential of semantic techniques, when the same video resources are exploited several times in different contexts of use (analyses using different domain ontologies, posting on different video channels).

This scenario involves the following users.

1. Video Material Owners (VMO) provide videos to the archive. They encrypt videos, upload them on CoNet and advertise the existence of the videos to specific analysts and video distributors. They want to be notified when their videos are manipulated.
2. Video Distributors (VD) are notified of new videos, download and decrypt them, then make the videos available for streaming.
3. Analysts describe and interpret the content of a video resource, i.e. through the analysis of audiovisual topics (or subjects), the analysis of visual and/or acoustic frames, the linguistic and cultural adaptation of source videos to a chosen (French speaking) target public, etc. They subscribe to videos, download them from CoNet and decrypt them. They upload their analyses on CoNet and notify certain VCOs of the existence of their analyses. They want to be notified when their analyses are manipulated.
4. Video Channel Owners (VCO) are responsible for video channels, and manage content posted on their channel. They subscribe to analyses and post them on their channel. They want to notify certain VCUs and CHs of these posts.
5. Channel Holders (CH) are notified of new channels and new posts on channels, update the channels and make them available on the Internet.
6. Video Channel Users (VCU) explore the content of Channels, subscribe to new posts on channels and browse channels.

5.2 Bird's eye view of the deployment framework

This paragraph describes the applications associated with each category of user are

a. VMO
    1. Gets Convergence identifier
    2. Subscribes to analyses related to his videos
    3. Receives notifications of analyses and posts
    4. Encrypts videos
    5. Identifies Videos
    6. Creates and sends Licenses for Videos
    7. Creates Video VDIs
    8. Identifies Video VDIs
    9. Stores Video VDIs in CoNet
    10. Publishes Videos as Publication VDIs
    11. Revokes P/S VDIs
    12. Revokes Video VDIs

b. VD
    1. Gets Convergence identifier
    2. Subscribes to videos
    3. Receives notifications of videos
    4. Makes videos available for streaming

c. Analyst
    1. Gets Convergence identifier
    2. Subscribes to videos
    3. Receives notifications of videos
    4. Gets video VDIs from CoNet
    5. Decrypts and stores videos
    6. Makes Analyses
    7. Creates Analysis VDIs
    8. Identifies Analysis VDIs
    9. Stores Analysis VDIs in CoNet
    10. Publishes Analyses as Publication VDIs
    11. Revokes P/S VDIs
    12. Revokes Analysis VDIs

d. VCO
    1. Gets Convergence identifier
    2. Subscribes to analyses
    3. Receives notifications of analyses
    4. Creates Channel VDIs
    5. Identifies Channel VDIs
    6. Stores Channel VDIs in CoNet
    7. Makes Posts of Analyses by updating Channels and creating new Channel VDIs
    8. Publishes Posts as Publication VDIs
    9. Revokes Posts of Analyses by updating Channels and creating new Channel VDIs
    10. Revokes P/S VDIs

e. CH
    1. Gets Convergence identifier
    2. Subscribes to Channels and Posts of Analyses
    3. Receives notifications of Channels and Posts of Analyses
    4. Creates and Identifies Channels
    5. Makes Channels available on the Internet

f. VCU
    1. Gets Convergence identifier
    2. Subscribes to Posts of Analyses
    3. Receives notifications of Posts of Analyses
    4. Creates and Identifies Channels
    5. Makes Channels available on the Internet

The scenario involves the following types of VDI:

1. Video VDI : represents a video resource
2. Analysis VDI: contains analytical appreciations of a video
3. Channel VDI: aggregates a set of Video Analysis VDIs
4. Publication VDI
5. Subscription VDI

5.3 Application Requirements

5.3.1 User registration and rights

5.3.1.1 User registration

User should provide an empty smart card with a PIN code (given to the user when he/she received the card), as well as the following personal information:

- First name
- Last name
- Email
- Company name

5.3.1.2 User rights

The table below, extracted from D.4.2 [3], summarizes the user rights in this scenario.

#	Issuer	Principal	Type of VDI	REL verbs	*Comments*
1	Analyst	Fractal	S-VDI of Video	Match, Notify	*Subscription to videos about a subject of interest*
2	FMSH	Fractal	S-VDI of Video	Match, Notify	*Subscription to videos from certain VMOs*
3	VMO	Analysts	R-VDI of Video	GovernedCopy, Play	*Analysts browse & download the video*
4	VMO	Fractal	P-VDI of Video	Match, Notify	*Publication of Video VDI*
5	VMO	FMSH (Video Distributor)	R-VDI of Video	GovernedCopy, Post (video)	*Video Distributor stores & stream the video*
6	VMO	VCO	R-VDI of Video	Play, Post (icon)	*VCOs browse video & post icon of video on their web channel*
7	VMO	Fractal	S-VDI of Analysis	Match, Notify	*Subscription of VMO to analyses of his videos*

8	VCO	Fractal	S-VDI of Analysis	Match, Notify	*Subscription to analyses about a subject of interest*
9	Analyst	VCO	R-VDI of Analysis	GovernedCopy, Post	*VCOs browse & download the analysis*
10	Analyst	Fractal	P-VDI of Analysis	Match, Notify	*Publication of Analysis VDI*
11	VCO	FMSH (Channel VDI Holder)	R-VDI of Channel	GovernedCopy, Post (Channel VDI)	*Channel VDI Holder stores and post channels*
12	VMO	VCU	R-VDI of Video	Play	*End-users watch video*
13	Analyst	VCU	R-VDI of Analysis	Play	*End-users read analysis*
14	VCO	VCU	R-VDI of Channel	Play	*End-users browse channel*
15	VCU	Fractal	S-VDI of Post	Match, Notify	*End-users subscribe to posts of analysis about a subject of interest*
16	VCO	Fractal	"P-VDI of Post of Analysis"	Match, Notify	*VCOs advertise the post of an analysis in their channel*

Table 5-1: User rights (REL verbs and conditions)

5.3.2 Security requirements

Security requirements for the scenario may be summarized as follows:

1. Users should be able to register to the system using a smart card
2. Users involved in the scenario should be able to digitally sign VDIs
3. A VMO should be able to encrypt his/her video and grant licenses with cryptographic key to users
4. Video encryption should support ABE (see section 5.3.2.1 below)
5. A licensed user should be able to receive his/her license including the cryptographic needed decrypt the relevant video resource
6. A remote client should be in position to authenticate himself /herself with his/her personal smart card.
7. The FMSH server should be in position to authenticate itself
8. FMSH should be able to create trusted groups of users and assign rights to users belonging to a particular group
9. An authorized user should be able to set a group of users as the principal of a license or as the recipient of an ERR
10. A user should be able to see the signature of a VDI

5.3.2.1 ABE (Attribute Based Encryption)

In addition to content encryption, as described in chapter 3, the FMSH scenario will also support Attribute Based Encryption (ABE, see Section 5.3 in D4.2 [3] for a brief primer on ABE). ABE allows the publisher of a video to encrypt the video so it can only be decrypted by recipients with certain specific attributes or logical combinations of attributes. In this way, a publisher can enforce access conditions phrased in terms of attributes without having to rely on a license-scheme (which recipients may or may not honour).

Examples of ABE expressions for downloading videos:

User	**Conditions on user characteristics**
FMSH Analyst	(belonging to group of FMSH-Analysts) AND (university level >= M1)
FMSH Video Distributor	(belonging to group of FMSH-VDs) AND (localized at FMSH premises in Paris, France)
INC Member	(belonging to group of INC-Members) AND (localized in Peru)
Peruvian government VCOS	(belonging to group of Peruvian Government-VCOs) AND (localized in Peru)
SUMMARY	
	((belonging to group of FMSH-Analysts) AND (university level >= M1) OR (belonging to group of FMSH-VDs) AND (localized at FMSH premises in Paris, France) OR (belonging to group of INC-Members) AND (localized in Peru) OR (belonging to group of Peruvian Government-VCOs) AND (localized in Peru))

Table 5-2: ABE Summary

The table below matches the security requirements described in paragraph 5.3.2 with the security functionalities defined in chapter 3 (abbreviations explained in the Glossary section).

Scenario functionality	**Enc_Key_Wrap**	**Dec_Key_Unwrap**	**Us_Reg_IP**	**Us_Reg_SP**	**Cl_Auth_Us_Pw**	**Cl_Auth_SC**	**Serv_Auth**	**Smard_Card_Role_Auth**	**Sig**	**Group_Sig**	**PKI**	**REL**
1			X	X								
2									X			
3	X											X
4	X											
5		X										X
6						X						
7							X					
8			X	X							X	
9												X
10									X			X

Table 5-3: Security requirements for the FMSH scenario

5.4 Application VDIs

In the first phase of the FMSH trials, CONVERGENCE will support the following functionality.

a. VMO
   1. Creates and stores a Video VDI
   2. Creates and injects a Publication VDI
   3. Revokes a Video VDI
   4. Revokes a Publication VDI

b. Analyst
   1. Creates and injects a Subscription VDI
   2. Downloads a Video

The scenario involves the following types of VDI

1. Video VDI : represents a video resource
2. Publication VDI
3. Subscription VDI

5.4.1 VMO creates a Video VDI

In the first application, a VMO wants to create a new Video VDI. Each such VDI, will contain metadata related either to the video (such as title, tags, date of capture, short description, authors, etc.) or to the producer. Additionally, the VMO will use the License Tool to set rights over the resource (according to Table 5-3: Security requirements for the FMSH scenario). After successful registration, he/she launches the Create Video VDI application. The procedure for creating a new R-VDI consists of the following steps:

1. *Initialize a new Resource VDI instance*
2. *Using the environment, the user fills the appropriate metadata (tags entry)*
3. *Add a video reference to the resource VDI*
4. *Set an expiry data for the VDI*
5. *Store the VDI*
6. *Advertise the VDI*

5.4.2 VMO creates and injects a Publication VDI

In the second application, a VMO creates and publishes a VDI, in the following steps:

1. *Initialize a new instance of the Publication VDI Tool, based on the R-VDI identifier*
2. *Set an expiry date for the VDI*
3. *Add the VDI to a specified fractal*
4. *Publish the VDI in the cloud*

5.4.3 Analyst creates and injects a Subscription VDI

In the third application, an Analyst wants to create a S-VDI and inject it into the cloud. This involves the following steps:

1. *Initialize a new instance of the Subscription VDI Tool*
2. *Set a fractal for the S-VDI*
3. *Set an expiry date for the S-VDI*
4. *Add the keywords and search tags specified by the user*
5. *Add an event report (an event is triggered on every match)*
6. *Publish the S-VDI*

5.4.4 VMO revokes or un-publishes VDI

In the fourth application, a VMO wants to revoke a Video VDI from CoNet or to un-publish a P-VDI from CoMid. The procedure is the same in both cases (see section Revoke VDI). The corresponding implementation code can be found in ANNEX A.

5.5 Integration with the middleware

5.5.1 Engines per application

The table below (Table 5-4: Engines per application) summarizes the list of TEs required by each application during the first phase of the trials. Table 5-5: Supported functionalities per trial shows the functionality to be supported in each phase of the trials.

Technology Engine	VMO creates and uploads a Video VDI	VMO creates and injects a Publication VDI	Analyst creates and injects a Subscription VDI	VMO revokes a Video VDI	VMO revokes a Publication VDI
Metadata	X	X	X		
REL	X	X	X		
ER	X				
Security	X	X	X	X	X
Overlay		X	X		
CoNet	X	X	X	X	X
CDS		X	X		
VDI	X	X	X		
MPEG-21 FF	X		X		

Table 5-4: Engines per application

Functionality	Phase 1	Phase 2	Phase 3
Username/ Password User Authentication	X	X	
Smart card supported User Authentication			X
Single User Identification	X	X	X
Group identity creation by FMSH Manager		X	X
Content Encryption		X	X
Content Decryption		X	X
ABE		X	X
Manually generated metadata for the creation of Video and Analysis VDIs	X	X	X
CDS-support generated metadata for the creation of Video and Analysis VDIs		X	X

Table 5-5: Supported functionalities per trial

6 Specifications and Technical Implementation of "Augmented Lecture Podcast (ALP)" (real world trial 3)

6.1 Brief description of use case

This scenario is based on a web-based lecture podcast application that enables students to revise lectures by watching video podcasts with synchronized slides. CONVERGENCE provides a common basis for collaboration and information exchange. Since different instances of the same file are related to each other by their VDI representations, it is possible to update one file and keep all related files in sync, ensuring that students always have access to the latest materials. Students who have downloaded files onto their local devices are notified of changes as soon as they connect to the CONVERGENCE Network. The scenario involves the following users:

a) Lecturers:
    a. Create, store and publish Lecture Podcast VDI sequences, which are updated whenever videos or slides are modified.
    b. Retrieve statistical information about the use of the Lecture Podcast VDIs (statistical information is collected by the service provider).

b) Students:
    a. Search, subscribe to, download and watch Lecture Podcasts.
    b. Receive notifications whenever the service makes available a new podcast or a new version of a podcast.
    c. Create and publish annotations to specific portions of the podcast.
    d. Receive notifications about published Annotations.
    e. Delete annotations.

6.2 Bird's eye view of the deployment framework

For the first phase of the trial the application space has been provisionally partitioned into six Tools and Applications.

1. User registration
2. Lecturers create and update video fragments and corresponding slides
3. Lecturers create and publish sequence of Lecture Podcast VDIs
4. Students search for slide/video/lecture podcast VDI
5. Students watch Lecture Podcasts, and write Annotations for a selected audience
6. Students read Annotations

6.3 Application Requirements

6.3.1 User Registration and rights

6.3.1.1 User Registration

Students can register for the ALP-Application to watch and annotate lecture podcasts. There are two registration mechanisms:

- Registration for smart cards. The following information is required:
  - Passport
  - Signature
- Registration via web interface. The following information is required:
  - First name and last name
  - Valid university account (email)
  - Password
  - User Information (Major / Minor / Semester/ …)

Lecturers can register to provide lecture podcast and teaching materials. There are two registration mechanisms:

- Registration for smart cards. The following information is required:
  - Passport
  - Signature
- Registration via web interface. The following information is required:
  - First name and last name
  - Valid university account (email)
  - Password

6.3.1.2 User Rights

The table below, extracted from D.4.2 [3], shows user rights for different categories of Principals and Rights

#	Issuer	Principal	Type of VDI	REL verbs	*Comments*
1	Student	Student	R-VDI of Annotation	Extend Rights, Play	*Display and usage of annotation*
2	Lecturers	Podcast Service	R-VDI	Post	*Provide learning materials to students*
3	Lecturers	Students	R-VDI for podcasts (components)	GovernedCopy, Play	*Download and Play/View slides and Video*

Table 6-1: User Rights for the ALP Application

6.3.2 Security requirements

1. Only students registered to the ALP-Service shall be allowed to make annotations in the ALP-Application
2. The author of the annotation shall define it to be:
   i. Public: visible to all students registered for the ALP-Service
   ii. Semi-private: visible to a group of students (specified by the author)
   iii. Private: only visible to the author

   To enforce these requirements, annotations are encrypted using key escrow encryption (a "fair" cryptosystem) [2].
3. Students shall be able to publish public annotations anonymously while remaining traceable under specific circumstances.
4. It must be clear (not necessarily visible) who the author of an annotation is.
5. If a user violates the terms of service, the group master shall be able to reveal the identity of that user (especially in the case of anonymous annotations).
6. Only lecturers shall be able to use the ALP-Creator application.
7. Only the creator of a podcast (component) VDI can retrieve its corresponding statistical information.

The table below matches the security requirements with the security functionalities defined in chapter 3 (abbreviations explained in the Glossary section).

[2] Key escrow (also known as a fair cryptosystem) is an arrangement in which the keys needed to decrypt encrypted data are held in escrow so that, under certain circumstances, an authorized third party may gain access to those keys. Access is permitted only under carefully controlled conditions, as for instance, a court order.

Scenario functionality	Enc_Key_Wrap	Dec_Key_Unwrap	Us_Reg_IP	Us_Reg_SP	Cl_Auth_Us_Pw	Cl_Auth_SC	Serv_Auth	Smart_Card_Role_Auth	Sig	Group_Sig	PKI	REL
1			X	X		X						
2	X	X										X
3			X	X						X		
4			X	X						X		
5										X		
6			X	X		X		X				X
7												X

Table 6-2: Security Requirements for the ALP Application

6.4 Creation of Application VDIs

A lecturer wants to create a podcast VDI of her lecture, which consists of video recordings with synchronized slides. She creates a new slide VDI and video VDI. In the final step, she creates a podcast VDI putting the VDIs together. Each VDI contains metadata related to the slides, video and podcast (e.g. synchronization information, duration, title, lecturer, term, description of podcast, etc.). In addition, the lecturer sets rights to her digital resources. These tasks are supported by the Augmented Lecture Podcast creator (ALP creator), which lecturers can launch after successful registration. Below, we describe the steps in the process:

1. *Creation of podcast components (slides, video recordings):*
    a. *Initialize a new Resource VDI instance*
    b. *Using the ALP creator, lecturers fill the appropriate metadata (tags entry)*
    c. *Add slides / video recordings to the resource VDI*
    d. *Store the VDI*
    e. *Advertise the VDI*

2. *Creating podcast VDI*
    a. *Repeat steps 1a and 1b*
    b. *Add a relationship between slide and video VDIs*
    c. *Store the VDI*
    d. *Advertise the VDI*

Students want to annotate the lecture podcast they are watching in their social learning environment, which is an augmented lecture podcast web application (ALP web application). For each annotation, a VDI is created, embedding the comment as a resource. The VDI also contains metadata (e.g. title, author…). This process involves the following steps:

1. *Initialize a new Resource VDI instance*
2. *Using information from the web application (e.g. user name, timestamp, etc.) tag entries are entered*
3. *Create a text file containing the comment*
4. *Add text file to the resource VDI*
5. *Store the VDI*
6. *Advertise the VDI*

The corresponding code can be found in ANNEX A.

6.4.1 Engines per application

The table below (Table 6-3: Engines per application) summarizes the list of TEs required for each application used in the first phase of the trials. Table 6-4: Supported functionalities per trial for the ALP scenario shows the functionality to be supported in each phase (L: Lecturer, S: Student, ALP: Augmented Lecture Podcast, LP: Lecture Podcast).

Technology Engine	L creates and uploads LP sequence	L creates and injects a publication VDI	L updates and revokes VDI	S creates and injects a Subscription VDI	S request podcast or podcast components/ annotations
	S creates/ stores annotation				
Metadata	X	X		X	
REL		X		X	
ER	X				X
Security	X	X	X	X	X
Overlay		X		X	X
CoNet	X	X	X	X	X
CDS					
VDI	X	X		X	X
MPEG-21 FF	X				

Table 6-3: Engines per application for the ALP scenario

6.4.2 Functionalities per trial

Functionality	Phase 1	Phase 2	Phase 3

Username / Password authentication	X	X	X
Smartcard supported user authentication			X
Manually generated metadata for describing slides / video / podcasts / annotations	X	X	X
CDS-support generated metadata for describing slides / video / podcasts / annotations	X	X	X
Content Encryption/Decryption		X	X
Keyword-based search		X	X
Semantic Search			X
Request slide/video/ podcast / annotation VDI		X	X
Annotations are public and non-anonymous	X	X	X
Annotations are public and anonymous			X
Annotations are private/semi-private		X	X
Annotation revocation		X	X
Event Report Browsing			X

Table 6-4: Supported functionalities per trial for the ALP scenario

7 Specifications and Technical Implementation of Smart Retailing (real world trial 4)

7.1 Brief description of use case

This scenario describes a smart retailing supply chain for electronic products. In the scenario, CONVERGENCE provides users with a wide range of services and operations. The users and beneficiaries are Manufacturers, Retailers and Customers. Manufacturers create their products VDIs and make them available by publishing them on the CONVERGENCE system. Retailers subscribe to products from Manufactures and advertise the products they sell with promotions, special offers, etc.. Customers subscribe to, search for and compare products. Additional services improve the Customer shopping experience allowing them to subscribe to information on specific products, receive notifications about the products and sales events and immediately look up information on a product by scanning its barcode at the Retailers' stores.

This scenario involves the following activities:

1. Manufacturers
    a. Create, store, publish and certify Product Type VDIs containing reliable information such as the name of the product, description, its physical and technical characteristics
    b. Profit from new flexible mechanisms for selling products to authorized Retailers in a smart retailing supply chain
2. Retailers
    a. Subscribe to Product Type VDIs published by Manufacturers
    b. Create and publish Retailers' Product Type VDIs, augmenting Manufacturer Product Type VDIs with information about Retailer promotions, special offers, sales, etc.
    c. Create Product Instance VDIs when selling products to Customers, adding some new information to the one present in the Product Type VDIs, such as customer information (id), serial number and warranty details
    d. Enjoy improved control over the products they sell and better knowledge of their Customers
3. Customers
    a. Search and subscribe to products VDIs
    b. Receive notifications about updates on products VDIs they have subscribed to
    c. Consult certified, reliable information about products
    d. Use services to manage technical information about their products, warranties, information on repair services etc.

7.2 Bird's eye view of the deployment framework

The scenario involves the following functionalities:

1. Manufacturer
    i. Create and Store Product Type VDI
    ii. Create and inject Publication VDI of the Product Type VDI

iii. Revoke Publication VDI from the CONVERGENCE Cloud

2. Retailer
   i. Create and inject Subscription VDI of Manufacturer's Product Type VDI
   ii. Download (extract metadata and resources) Manufacturer's Product Type VDI
   iii. Create and inject Publication VDI about promotions, special offers on Product Type VDIs
   iv. Process Product Type VDI to Create and Store Product Instance VDI (adding new information)
   v. Revoke Publication VDI from the CONVERGENCE Cloud
3. Customer
   i. Create and inject Subscription VDI
   ii. Browse products VDIs

7.3 Application Requirements

7.3.1 User registration and rights

7.3.1.1 User registration

Users register to their CONVERGENCE applications by choosing a username and a password. They can also provide additional personal information, including email, company name, etc. After the registration process is complete the user receives a smartcard with his/her certificate and other credentials used for purposes of authentication.

7.3.1.2 User rights

The table below, extracted from D.4.2 [3], describes the rights assigned to different categories of user.

#	Issuer	Principal	Type of VDI	REL verbs	*Comments*
1	Manufacturer	Retailer	R-VDI of Product Type	GorvernedAdapt	*Add information to a Product Type VDI*
2	Manufacturer	Retailer	R-VDI of Product Type	Play, Print	*Display image (photo) or document (user manual) of product* *Print a photo or document related to the product*
3	Manufacturer	Fractal	P-VDI of Product Type	Match, Notify	*Publication of Product Type VDIs*
4	Retailer	Fractal	S-VDI of Product Type	Match, Notify	*Subscription of Product Type VDIs from Manufacturer*

5	Retailer	Fractal	P-VDI of Product	Match, Notify	*Publication of Product VDI about promotions, special offers, etc.*
6	Retailer	Customer	R-VDI of Product	Play, Print	*Customers browse products VDIs and play/print the resources (photos or documents)*
7	Retailer	Customer	R-VDI of Product Instance	GovernedCopy, Adapt, Delete	*Create a Product Instance VDI linked to the Product Type VDI*
8	Customer	Fractal	S-VDI of Product	Match, Notify	*Subscription to products VDIs*

Table 7-1: User Rights for the Smart Retailing scenario

7.3.2 Security Requirements

1. Only registered users (Manufacturers and Retailers) should be able to create product related VDIs.
2. Users responsible for the creation of product related VDIs shall be able to sign them.
3. Manufacturers shall be able to identify and certify the official Retailers responsible for selling their products to Customers.
4. Manufacturers shall be able to protect VDI information for their products, preventing unauthorized modifications including modifications by official Retailers or Customers.
5. The information visible in a product VDI should be defined by owner of the VDI.

The table below matches the security requirements with the security functionalities defined in chapter 3 (abbreviations explained in the Glossary section).

Scenario functionality	**Enc_Key_Wrap**	**Dec_Key_Unwrap**	**Us_Reg_IP**	**Us_Reg_SP**	**Cl_Auth_Us_Pw**	**Cl_Auth_SC**	**Serv_Auth**	**Smart_Card_Role_ Auth**	**Sig**	**PKI**	**Group_Sig**
1			X	X	X	X					
2									X		
3			X	X						X	
4									X		
5								X			X

Table 7-2: Security requirements for the Smart Retailing scenario

7.4 Creation of Application VDIs

7.4.1 Manufacturer Create and Store Product Type VDI (R-VDI)

A Manufacturer wants to create a VDI for a new product he is releasing to the market. He uses the application to create and store a product type VDI containing product information (metadata) and product resources, such as photos and user manuals. After the product type VDI is created, the application advertises it. This involves the following steps:

1. *Initialize a new Resource VDI instance*
2. *Using the web application manufacturers fill the Product VDI Creator form with the product information that will be used as metadata (tags entry)*
3. *Add files (url, mime type) to the Resource VDI. For example: product photos, user manuals.*
4. *Store the product type R-VDI*
5. *Advertise product type R-VDI*

7.4.2 Manufacturer Publish Product Type VDI (P-VDI)

To make the product type VDI available to Retailers, the manufacturer uses the application to publish the VDI, associating a fractal and an expiry date with the publication, ad optionally adding an event report. This involves the following steps:

1. *Initialize a new Publication VDI instance*
2. *Using the web application manufacturers fill the Publish Product form with the fractal and expiration date of the publication which will be added to the Publication VDI variable*
3. *Add the event report to the Publication VDI*
4. *Publish the PVDI*

7.4.3 Retailer Subscribe Product Type VDI (S-VDI)

A Retailer wants to search for product type VDIs with certain features or characteristics. To do that he uses the application to subscribe to all VDIs, in a particular fractal, that meet his search criteria. This involves the following steps

1. *Initialize a new Subscription VDI instance*
2. *Using the web application retailers fill the Subscribe to Product form with the fractal and expiration date of the publication which will be added to the Subscription VDI variable*
3. *Add condition to the Subscription VDI*
4. *Add event report to the Subscription VDI*
5. *Publish the S-VDI*

7.4.4 Retailer Creates Product Instance VDI (R-VDI)

This scenario involves the POS machine used by the Retailer when selling a product to a Customer. The machine identifies the product that is being sold to the Customer and gets the

product type VDI associated with it. Then the Retailer inputs information about the product serial number and warranty detail. The Customer presents his personal information to the Retailer, either by inputting it manually into the POS or by presenting his/her loyalty card. This information now becomes metadata in the new product instance VDI, representing the product sold to the Customer. Finally, the system creates a link between the product type VDI and the created product instance VDI. The process involves the following steps:

1. *Initialize a new Resource VDI Tool*
2. *Read the product type Resource VDI instance and get its metadata*
3. *Initialize a new Resource VDI instance*
4. *Add the metadata of step 2*
5. *Using the POS client application retailers input the product instance serial number and the customer information that will be used as metadata (tags entry)*
6. *Add the digital receipt file and the warranty details file (url, mime type) to the Resource VDI.*
7. *Add the link between the product type R-VDI and the product instance d*
8. *Store the product instance R-VDI*
9. *Advertise the product instance R-VDI*

7.5 Integration with the middleware

7.5.1 Engines per application

Technology Engine	Creation and storage of product VDI	Creation and injection of Publication VDI	Creation and injection of Subscription VDI
Metadata TE	X	X	X
REL TE	X	X	X
ER TE		X	X
Security TE	X	X	X
Overlay TE		X	X
CoNet TE	X	X	X
CDS TE	X	X	X
VDI TE	X	X	X
MPEG-21 FF	X		X

Table 7-3: Engines per application for the Smart Retailing scenario

7.5.2 Functionalities per trial phase

Functionality	Phase 1	Phase 2	Phase 3
Username/ Password User Authentication	X	X	

Smart card supported User Authentication			X
Single User Identification	X	X	X
Group identity creation by Manufactures		X	X
Manually generated metadata for the creation of Product VDIs	X	X	X
CDS-support generated metadata for the creation of Product VDIs		X	X
Creation and injection of Publication VDI	X	X	X
Creation and injection of Subscription VDI	X	X	X
Revoke VDI			X
Browse products VDIs		X	X
Browse ERR			X

Table 7-4: Supported functionalities per trial for the Smart Retailing scenario

8 Summary - future work

This deliverable describes the implementation and security framework for the applications to be deployed in the first CONVERGENCE trial. The information provided includes:

- The definition of Tool APIs.
- The definition of security functionality for content protection (e.g. encryption/decryption) and for secure transactions (e.g. usage of smart cards and manipulation of digital signatures) (see ANNEX B for a detailed description of APIs)
- Procedures for the creation of the R-VDIs, P-VDIs and S-VDIs required for the four user scenarios

In chapters 4-7 we also described the use of:

- Smart cards, to support user authentication and registration.
- CDS – supported metadata, with specific ontologies for different applications.
- Group identity, as a means of creating "trusted" groups of users (e.g. photographers, analysts, students, retailers).
- Extension of REL, to support group licensing.

We expect to complete the implementation of these functions before the start of the third phase of the trials.

In parallel with this work, the second track of the trials (not discussed in the earlier chapters of this deliverable) will perform network experiments, investigating how features of CONET, such as content-routing and in-network caching, can contribute to the performance of CONVERGENCE applications. This topic is briefly discussed in the next section, and gives a partial answer to **some comments raised during the first year review**.

8.1 User scenarios involving CONET (Track 2)

In the second track of the trials we plan to setup a couple of simple scenarios which show how CONVERGENCE application may exploit the features of CONET, i.e. of an "Information Centric" Network (ICN).

We argue that two of the most important features of CONET are in-network caching and content routing; therefore in track 2 we envisage two challenging scenarios where these features strongly improve user experience.

In-network caching allow network nodes to cache chunks of data (i.e. name-data CIUs), so improving content delivery in cases of a spatial locality of user requests. Content-routing allows network node to route data requests toward the more convenient serving node.

In what follows, we describe two possible scenarios of track 2 where we plan to reuse CoMID TEs and applications of "Photos/Videos in the Cloud and Analyses down to earth" developed for track 1. In the first scenario of track 2 we focus on the use of in-network caching in a mobile environment. In the second scenario of track 2 we focus on a fixed environment where we show the use of both in-network caching and content routing for a video streaming application. We observe that mobile applications and video streaming are two very relevant

use-cases for a Future Internet networking technology. Therefore, showing the use of CONET in these scenarios is a fundamental step to assess its effectiveness.

8.1.1 Shared museum

Figure 8-1 shows the first trial scenario we considered for track 2, named “shared museum”. We have a set of neighbour users visiting a monument. By using their mobile devices, the users wish to have historical information on the monument. To this aim the users use a CONVERGENCE museum application that is similar to the one we implement within the “Photos in the Cloud and Analyses down to earth” track 1 trial. Nevertheless, while in track 1 the application and COMID Engines are based on a plain TCP/IP networking layer, in track 2 they are supported by the CONET layer.

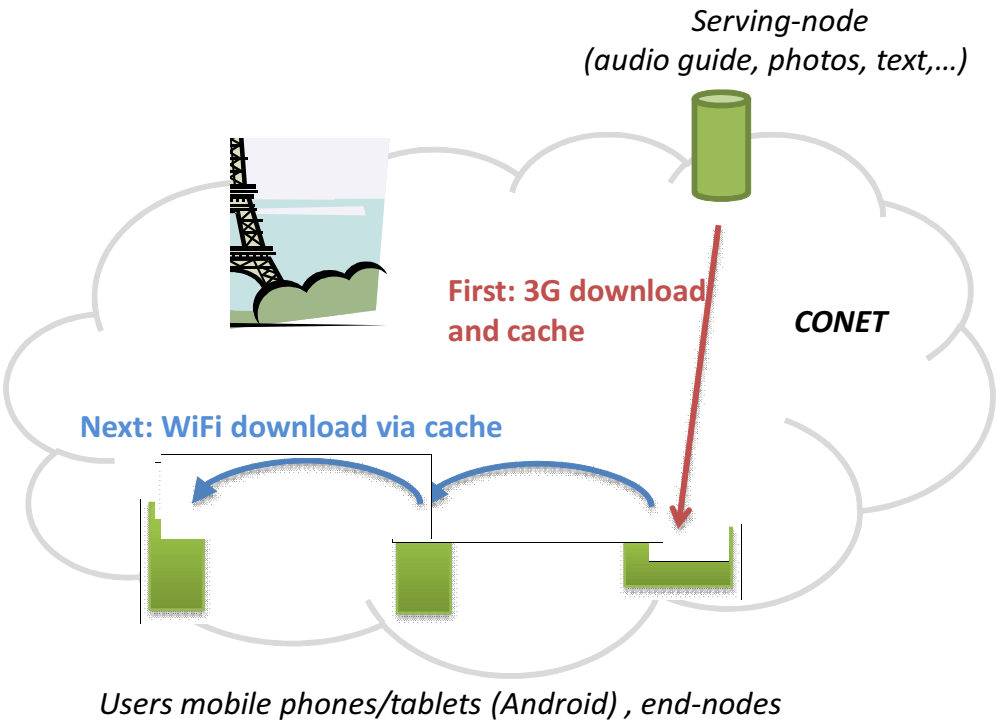

Figure 8-1: Shared museum

In this scenario there are many users in the same location searching for similar content; thus, this scenario is very suitable to show the potential effectiveness of in-network caching. To this aim, we envisage that a first user downloads a VDI and related resources (e.g. an audio-guide with photos and text) from an origin serving-node via 3G. After the download, the mobile device (a CONET end-node) caches the downloaded data. Other users that later on will search the same VDI/resources will be able to download them via WiFi from the cache of neighbour users.

Generalizing the scenario, we have a set of mobile devices that form a WiFi ad-hoc network (single hop) while at the same time can exploit a 3G connection. When a user wishes to download a resource, the CONET layer tries to download the resource from the caches of neighbour devices. In case of cache miss, the CONET layer downloads the resource via 3G

from the origin serving-node. After the download, the CONET layer caches the resource in the mobile device.

We observe that in this scenario in-network caching is useful both for the user and for the 3G mobile operator.

Many users may download resources from the local WiFi ad-hoc network, enjoying a fast delivery and saving money, in case the 3G operator applies a volume-based pricing model. Moreover, foreign users may not like to use 3G, to avoid high roaming costs. Using CONET these users may download the desired content via WiFi.

Mobile operators may also benefit from the CONET. Indeed, especially in case of a flat rate pricing model, the 3G interface would be off-loaded.

We finally observe that, as occur in P2P file sharing, suitable policies/credit-systems should be deployed to discourage "free-riders" (users that download only), and to attract users to participate to the application.

8.1.2 Video streaming for small businesses

Figure 8-2 shows the second scenario we consider for track 2, named "Video streaming for small businesses". We consider a SME wishing to sell at a given time (e.g. 21:00 PM) a HD video streaming of a recent movie. Users exploit an application similar to the one we implement in the track 1 trial of "Videos in the Cloud and Analyses down to earth", properly adapted to exploit the CONET layer.

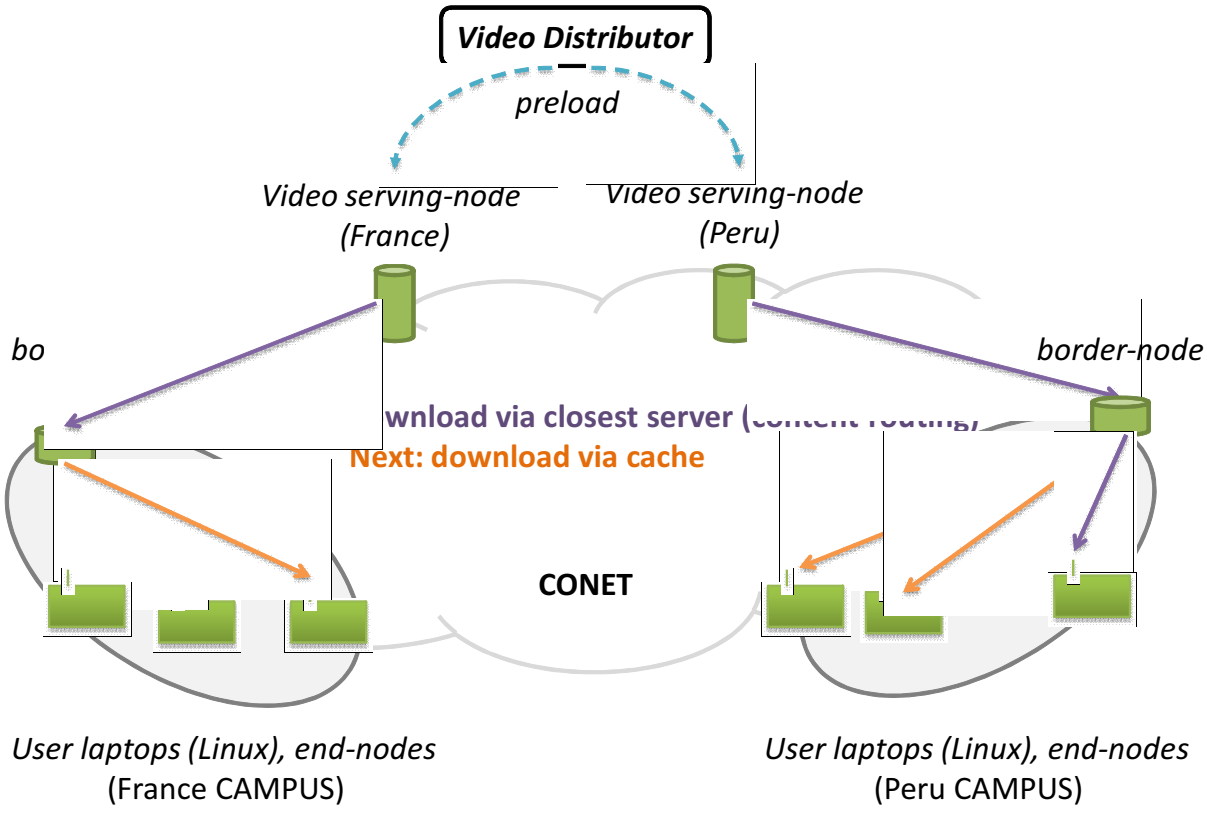

Figure 8-2: 8.1.2 Video streaming for small businesses

We assume that the small price of the video prevents the video distributor to pay a CDN provider, which could massively replicate video servers around the world (as it occurs for iTunes, for instance). In the current Internet, this scenario would not be possible: the video could not be streamed to a large number of users without the support of a CDN, which would not be possible given the expected, limited revenue of this business. In the scenario reported in Figure 8-2 we show how this impasse can be overcome by using CONET, and in particular the content-routing and in-network caching features.

The video distributor deploys a small set of video servers (small with respect to typical CDN scale). On these serving-nodes, the video distributor preloads the VDI and the movie to stream. The streaming format is such that video frames are grouped in chunks and each chunk has a different Network Identifier (NID). Therefore, each chunk is handled by the CONET as a named-data.

Let us assume that the video is successful and that at the start of the streaming (e.g. 21:00 PM), a large number of users download the VDI and join the streaming session[3]. In what follows we restrict our focus on the users of the Peru and France university campuses. Nevertheless, the same scenario could be applied to the clients of an ISP.

The first user of the campus issues a request (Interest CIU) for chunk n.1 of the video. The chunk request is routed towards the CONET border-node of the campus, which could be located where today the IP default gateway is located. The border-node forwards-by-name (aka content-routing) this request to the nearest video server. Content-routing provides also load balancing among the small set of video servers of the video distributor. When the video server sends back chunk n.1, the border-node caches it. Next users of the same campus that request chunk n.1 will retrieve it from the cache of the border-node. The same happens for the other chunks of the video. Therefore, only a single stream per-campus is downloaded from the video server, while other streams are served from the cache. In this way, the video server is significantly unloaded and a large number of users can watch the video.

8.2 Other advantages of Convergence and benefits for the four main real world trials/scenarios

The two scenarios described above can be loosely put in relationships with the first two main real world trials/scenarios of Convergence, namely “Photos in the Cloud and Analyses down to earth” and “Videos in the Cloud and Analyses down to earth”.

As regards the third main real world trials/scenarios of Convergence, namely Augmented Lecture Podcast, we note that the CoNet brings advantages not only in allowing caching but also in reassembling and using learning content. Lecturers do not have to explicitly ask for permission to each author to use content (who may have asked permission from other authors as well), since CONVERGENCE works with self-consistent packages, including the definition of licenses and REL. For students, this will mean content integrity and reliable lecture materials.

As regards the fourth main real world trials/scenarios of Convergence, namely Smart Retailing, we note that CoNet brings advantages not only in allowing caching and content routing but also in allowing to digitally sign and encrypt a resource facilitating very much retailing procedures and providing a strong security mechanism against, for example, frauds

[3] To support late joining, VDIs can include the NID of the next video chunks where joining is possible, being the movie actually made of a sequence of VDIs.

on product returns. Additionally, in cases of “safety recall” the author of a VDI can send a message to all users owning a copy of that VDI, without knowing their personal details, and allowing the owner of a VDI to authorize or refuse reception of such messages. Finally, named content is expected to significantly reduce search and retrieve content time once a customer is inside the store. As nowadays the globalization of the electronic market supply chain is a fact, product characterization becomes a major issue. The VDI technology could help in describing and advertising products.

Finally, we note that ICN in general and CoNet in particular could be a great facilitator for Cloud services, thus benefiting a large set of possible applications, including the selected Convergence ones.

9 References

[1] CONVERGENCE Project Deliverable D.7.1

[2] CONVERGENCE Project Deliverable D.8.1

[3] CONVERGENCE Project Deliverable D.4.2

10 ANNEX A

In this ANNEX the code describing the creation of the individual applications is provided.

10.1 Photographer creates a Photo VDI

- ResourceVdiTool rvdiTool = new ResourceVdiTool ();
  Initializes a new Resource VDI instance
- rvdiTool.addTag(tag);
  In the GUI environment, the user fills the appropriate metadata (tags entry)
- rvdiTool.addResource(url, "image/" + photo.getImageFormat());
  Adds to the resource VDI an image
- rvdiTool.setExpiryDate(new Date());
  Sets an expiry data to the VDI
- rvdiTool.store(filePath);
  Stores the VDI to the specific filepath
- rvdiTool.advertise(filePath);
  Advertises the VDI

10.2 Photographer publishes Photo VDI

- PublicationVdiTool pubVdiTool = new PublicationVdiTool(photo.getR_VDI_Id());
  A new instance of the publication tool is created
- pubVdiTool.setExpiryDate(finalDate.getTime());
  An expiry date is set to the P-VDI
- pubVdiTool.setFractal(fractal);
  The P-VDI is added to a specific fractal
- pubVdiTool.generatePVDI();
  The P-VDI is generated
- pubVdiTool.publish();
  The P-VDI is published

10.3 Alinari Photo Archive Manager subscribes to and downloads photos

- ERR err = EventReportTool.getInstance().createERR(EventTriggerType.MATCH);
  An event is triggered for every match
- SubscriptionVdiTool svdiTool = new SubscriptionVdiTool();
  A new instance of the subscription tool is created
- svdiTool.setFractal(request.getParameter("fractal"));
  A search fractal is set to the S-VDI
- svdiTool.setExpiryDate(finalDate.getTime());
  An expiry date is set to the VDI

- svdiTool.addKeywordCondition(tags);
  The search tags defined by the user are passed to the VDI
- svdiTool.addERR(err);
  The event report is added
- svdiTool.generateSVDI();
  The S-VDI is generated
- String svdi_id = svdiTool.publish();
  The S-VDI is published in CoMid

10.4 VMO creates a Video VDI

- ResourceVdiTool rvdiTool = new ResourceVdiTool ();
  Initializes a new Resource VDI instance
- rvdiTool.addTag(tag);
  In the GUI environment, the user fills the appropriate metadata (tags entry)
- rvdiTool.addResource(url, "video/" + video.getImageFormat());
  Adds to the resource VDI an image
- rvdiTool.setExpiryDate(new Date());
  Sets an expiry data to the VDI
- rvdiTool.store(filePath);
  Stores the VDI to the specific filepath
- rvdiTool.advertise(filePath);
  Advertises the VDI

10.5 VMO creates and injects a Publication VDI

The code is identical to the code for the Alinari scenario described in section 10.2.

10.6 Analyst creates and injects a Subscription VDI

The code is identical to the code for the Alinari scenario described in section 10.3.

10.7 Lecturer creates podcast VDI

Creation of podcast components (slides, video recordings):

- ResourceVDITool rvdiTool = new ResourceVDITool ();
  Initialize a new Resource VDI instance
- rvdiTool.addResource (filepath, “video/mp4”); or
  rvdiTool.addResource(filepath, “application/pdf”);
  Add slides / video recordings to the resource VDI
- rvdiTool.addTag(tag);
  Lecturers use the ALP creator to fill in the appropriate metadata (tags entry)
- rvdiTool.store(filepath);
  Store the VDI to the specific filepath
- rvdiTool.advertise(filepath);

Advertise the VDI

Creating podcast VDI

- ResourceVDITool rvdiTool = new ResourceVDITool ();
  Initialize a new Resource VDI instance
- rvdiTool.addRelationship(relationshipType, slideVdiId)
  Add a relationship between slide and video VDIs
- rvdiTool.store(filepath);
  Store the VDI to the specific filepath
- rvdiTool.advertise(filepath);
  Advertise the VDI

10.8 Students create annotation VDI

- AnnotationVdiTool avdiTool.getInstance();
  Initialize a new Annotation VDI instance
  FileWriter fw = new FileWriter(file);
  (new PrintWriter(fw)).println(comment);
  Create a text file containing the comment
- avdiTool.addComment(null, file.getAbsolutPath(), “text”);
  Add text file to the resource VDI
- avdiTool.addFieldValue(field, value);
  Using information from the web application (e.g. user name, timestamp, etc.)
- avdi.store(filepath);
  Store the VDI to the specific filepath
- avdi.advertise(filepath);
- *Advertise the VDI*

10.9 Manufacturer creates Product Type VDI

- ResourceVdiTool rvdiTool = new ResourceVdiTool();
  Initializes a new Resource VDI instance
- rvdiTool.addResource(url_resource, product.getResourceFiletype());
  Adds to the resource VDI files (image or document)
- rvdiTool.setExpiryDate(new Date());
  Sets an expiry data for the VDI
- rvdiTool.addKeyword(tag);
  Add keywords (tag entry) to the VDI about the product
- rvdiTool.store(filePath);
  Stores the VDI to the specific filepath
- rvdiTool.advertise(filePath);
  Advertises the VDI

10.10 Manufacturer publishes Product Type VDI

The code is identical to the code for the Alinari scenario described in section 10.2.

10.11 Retailer subscribes to product

The code is identical to the code for the Alinari scenario described in section 10.3.

10.12 Retailer publishes Product Type VDI

Basically the same as 10.7 plus 2 additional calls:

- pubVdiTool.addKeyword()

Search tags defined by the publisher are passed to the VDI

- pubVdiTool.addFieldValue("price", product.getPrice())

Field – value metadata defined by the publisher are passed to the VDI

10.13 Retailer creates Product Instance VDI

- ResourceVdiTool rvdiTool = new ResourceVdiTool(rvdi);
  Initializes a new Resource VDI instance
- rvdiTool.getFieldValueMap();
  The metadata (tags entry) is extracted from the product type VDI, including also the serial number of the product, customer information and warranty details
- rvdiTool = new ResourceVdiTool();
  Initializes a new Resource VDI instance
- rvdiTool.addResource(url_resource, product.getResourceFiletype());
  Adds to the resource VDI the receipt and warranty files
- rvdiTool.addFieldValue(field, value);
  Adds the previously created metadata to the VDI
- rvdiTool.store(filePath);
  Stores the VDI to the specific filepath
- rvdiTool.advertise(filePath);
  Advertises the VDI

10.14 Consumer subscribes to product

- ERR err = EventReportTool.getInstance().createERR(EventTriggerType.MATCH);
  An event is triggered for every match
- SubscriptionVdiTool svdiTool = new SubscriptionVdiTool();
  A new instance of the subscription tool is created
- svdiTool.setFractal(request.getParameter("fractal"));
  A search fractal is set to the S-VDI
- svdiTool.setExpiryDate(finalDate.getTime());
  An expiry date is set to the VDI

- svdiTool.addKeywordCondition(tags);
  The search tags defined by the user are passed to the VDI
- svdiTool.addFieldValueCondition("price", "op",valuePrice);
  Adds the field value condition on product price
- svdiTool.addERR(err);
  The event report is added
- svdiTool.generateSVDI();
  The S-VDI is generated
- String svdi_id = svdiTool.publish();
  The S-VDI is published in CoMid

11 ANNEX B Brief description of the CoSEC API

11.1 Certificate Manager Interface

Classes implementing this interface are responsible for the generation of asymmetric key pairs (both on- and off-card), as well as the generation and verification of dedicated digital certificates.

Operation	Description
MXMObject **generateRSAKeyPair** (String *keylength*, String *PubExp*) throws CertificateManagerException	Generate an RSA key pair (a modulus, a private key and associated public key) **Parameters:** *keylength*: length of RSA modulus to be generated for this key pair (supported in first round: RSA 1024/2048) *PubExp:* designated public exponent **Action:** generates an RSA key pair **Returns:** keyIdentifier – randomly generated identifier public key: modulus, public exponent private key: private exponent
MXMObject **generateRSAKeyPairCard** (Byte *keyIdentifier*, String *keylength*, String *PubExp*) throws CertificateManagerException	Generate an RSA key pair (a modulus, a private key and associated public key) on the smart card **Parameters:** *keyIdentifier:* key identifier (for referencing the corresponding private key on card) *keylength*: length of RSA modulus to be generated for this key pair *PubExp*: designated public exponent (supported in first round: RSA 1024/2048) **Action:** generates an RSA key pair, internally validates possession of private key **Returns:** public key: modulus, public exponent **[Note:** *the private key is not returned and will never leave the card.*]
String **generateCVCertificate** (String *holderReference*, String *issuerReference*, String *expiryDate*, String *pubKey*, String *SignOID*)	This method is used to generate a card-verifiable certificate on a public key (belonging to a public-private key pair which has been generated off card) **Parameters:**

throws CertificateManagerException	*holderReference*: the identifier of the user owning the key pair *issuerReference*: the identifier of the certificate issuer *expiryDate*: date when the certificate expires *pubKey*: the public key of the holder to be certified (shall contain information on the key/algorithm type through an object identifier) *SignOID*: object identifier encoding the algorithm type to be selected for the certificate signature **Action:** signs and issues a card-verifiable certificate **Returns:** the signed card-verifiable certificate
String **generateCertificate** (String holderReference, String issuerReference, String expiryDate, String pubKey, Byte keyIdentifier, String SignOID) throws CertificateManagerException	This method is used to generate a certificate on a public key (belonging to a public-private key pair which has been generated on card) **Parameters:** *holderReference*: the identifier of the user owning the key pair *issuerReference*: the identifier of the certificate issuer *expiryDate*: date when the certificate expires *pubKey*: the public key of the holder to be certified (shall contain information on the key/algorithm type through an object identifier) *keyIdentifier*: key identifier (for referencing the corresponding certificate on card) *SignOID*: object identifier encoding the algorithm type to be selected for the certificate signature **Action:** signs and issues a certificate **Returns:** the signed certificate

11.2 Key Manager Interface

Classes implementing this interface are responsible for providing a number of security-related functionalities such as the generation of keys for symmetric cryptographic algorithms, symmetric encryption and decryption, and wrapping/unwrapping of encryption keys.

Operation	Description
byte [] org.iso.mpeg.mxm.engine.securityengine.KeyManager.**wrap**(byte[] *symmKey*, String *algorithm*, String *keyAlias*, String *certificate*) throws KeyManagerException	This method is used to asymmetrically encrypt ("wrap") a symmetric decryption key with a given algorithm using the public key from a certificate identified by a keyAlias. **Parameters:** *symmKey*: the key to be wrapped. This key will be needed for decryption of data encrypted with a symmetric cipher

	algorithm: the algorithm to be used to wrap the key (supported in the first trials: RSA 1024/2048) *keyAlias*: the alias associated to the public key used to wrap *certificate*: a certificate associated to (and containing) the public key to be used to encrypt the data **Actions:** validates the certificate and extracts the public key associated to *keyAlias*, then wraps *symmKey* **Returns:** the wrapped (asymmetrically encrypted) encryption key as a byte array (including an identifier carrying information on the *algorithm* used for wrapping) **Exceptions:** *certificateFailure* – the certificate could not be validated
byte [] org.iso.mpeg.mxm.engine.securityengine.KeyManager.**unwrap** (byte[] *wrappedsymmKey*, String *keyAlias*, String *PIN*) throws KeyManagerException	This method is used to asymmetrically decrypt ("unwrap") the key in input using the private key associated to a certain key alias given in input. Unwrapping will be performed with the private key (belonging to the key pair associated to *keyAlias* designated for asymmetric encryption/decription) on the user's smart card. [**Note:** the user's private key will never leave the smart card.] **Parameters:** *wrappedsymmKey*: the wrapped key to be unwrapped *keyAlias*: the alias associated to the key to be used to unwrap the wrapped symmetric key in input *PIN*: the smart card's PIN to access the unwrap functionality **Actions:** the wrapped key (usually included in a license) is transmitted to the smart card for unwrapping on-card **Returns:** the unwrapped decryption key as a byte array **Exceptions:** KeyManagerException
byte [] org.iso.mpeg.mxm.engine.securityengine.KeyManager.**encrypt** (byte[] *data*, byte[] *encryptionKey*, String *algorithm*) throws KeyManagerException	This method is used to symmetrically encrypt the data in input with a given symmetric key given in input. **Parameters:** *data*: the data to be encrypted *encryptionKey*: the key to be used to encrypt the data *algorithm* - the algorithm to be used to encrypt the data

	Returns: the encrypted data as a byte array **Exceptions:** KeyManagerException
byte [] org.iso.mpeg.mxm.engine.securityengine.KeyManager.**decrypt** (byte[] *data*, byte[] *decryptionKey*, String *algorithm*) throws KeyManagerException	This method is used to symmetrically decrypt the data in input with a given symmetric key given in input. **Parameters:** *data*: the data to be encrypted *decryptionKey*: the key to be used to decrypt the data *algorithm*: the algorithm to be used to encrypt the data **Returns:** the decrypted data as a byte array **Exceptions:** KeyManagerException
byte [] org.iso.mpeg.mxm.engine.securityengine.KeyManager.**generateRandomKey** (String *algorithm*, int *keyLength*) throws KeyManagerException	Generate a random key using a specific algorithm. **Parameters:** *algorithm*: the algorithm used to generate the random key *keyLength*: the length of the key to be generated, in bytes. **Returns:** the generated key (fulfilling the conditions of the specified algorithm) **Exceptions:** KeyManagerException

11.3 Signature Manager Interface

Classes implementing this interface are responsible for providing a number of functionalities related to digital (asymmetric) signatures, such as the generation and verification of hash-values, issuance and verification of digital signatures.

byte [] org.iso.mpeg.mxm.engine.securityengine.SignatureManager.**generateHash** (byte[] *data*, String *algorithm*) throws SignatureManagerException	This method generates a hash value of the data in input using a given algorithm. **Parameters:** *data* : the data to be hashed *algorithm*: the algorithm used to calculate a hash value of the data **Returns:** the hash value **Exceptions:**

	Signature Manager Exception
byte [] org.iso.mpeg.mxm.engine.securityengine.SignatureManager.**generateHash** (FileInputStream *fis*, String *algorithm*) throws SignatureManagerException	This method generates a hash value of the data in input using a given algorithm. **Parameters:** *fis*: the FileInputStream of the file containing the data to be hashed *algorithm*: the algorithm used to calculate a hash value of the data **Returns:** the hash value **Exceptions:** SignatureManagerException
boolean org.iso.mpeg.mxm.engine.securityengine.SignatureManager.**verifyHash** (byte[] *data*, byte[] *hashValue*, String *algorithm*) throws SignatureManagerException	This method verifies that the hash value of the data in input has a specific value. **Parameters:** *data*: the data on which the hash value has to be calculated *hashValue*: the hash value which has to be verified against the one calculated *algorithm*: the algorithm used to calculate the hash value of the data **Returns:** true if the calculated value is equal to the generated one **Exceptions:** SignatureManagerException
byte [] org.iso.mpeg.mxm.engine.securityengine.SignatureManager.**generateCardSignature** (byte[] *message*, String *algorithm*, String *keyAlias*, String *PIN*) throws SignatureManagerException	This method is used to generate a digital signature for the data in input using the given algorithm and a private key associated to a key alias. The signature is generated on the user's smart card. **Parameters:** *message*: the data to be signed *algorithm*: the algorithm used to sign the data *keyAlias*: the alias associated to the private key to be used to sign the data *PIN*: the password to authorize the signature operation on the smart card repository **[Note:** Internally, this method splits up signature generation into (a) partial hashing and (b) final hashing + signature generation. Partial hashing (of a potentially large message) may be performed on the user's PC. Final hashing and the actual signature generation involving the user's private signature key is performed

	on the smart card. The private signature key is never released from the card to the outside world.] **Returns:** a digital signature on the input message **Exceptions:** SignatureManagerException
byte [] org.iso.mpeg.mxm.engine.securityengine.SignatureManager.**generateSignature** (byte[] *message*, String *algorithm*, String *keyAlias*, String *password*) throws SignatureManagerException	This method is used to generate a digital signature for the data in input using the given algorithm and a private key associated to a key alias. The signature is generated off-card on a PC. **Parameters:** *message* : the data to be signed *algorithm*: the algorithm used to sign the data *keyAlias*: the alias associated to the private key to be used to sign the data *password*: the password to access the key repository **[Note:** the key repository will usually be a software program, but might also be a Hardware Secure Module] **Returns:** a digital signature on the input message **Exceptions:** SignatureManagerException
boolean org.iso.mpeg.mxm.engine.securityengine.SignatureManager.**verifySignature** (byte[] *message*, byte[] *signature*, String *algorithm*, String *keyAlias*) throws SignatureManagerException	This method is used to verify a signature on the message in input. [**Note:** it is advisable to obtain and validate a certificate on the issuer's public signature key prior to the verification process.] **Parameters:** *message*: the data upon which the digital signature has been generated *signature*: the signature to be verified *algorithm*: the algorithm with which the signature has been generated *keyAlias*: the alias associated to the public key to be used to verify the signature **Returns:** true if the verification of the digital signature succeeds, false otherwise **Exceptions:** SignatureManagerException

11.4 Authentication Interface

Classes implementing this interface are responsible for providing functionalities related to client and server authentication deploying users' smart cards as secure repositories. Currently, methods are integrated supporting two authentication protocols (namely server authentication and client (user) authentication).

String **getCardChallenge**() throws AuthenticationManagerException	Part of server authentication towards a client based on RSA. Retrieves a random challenge from the client's smart card. The retrieved challenge shall then be sent to the remote server for signing. This method is invoked on the client connected to the user's smart card. **Returns:** A random challenge retrieved from the smart card.
Boolean **verifyCVCertificate**(String *CVcertificate*) throws AuthenticationManagerException	Part of server authentication towards a client based on RSA. Passes a certificate retrieved from the server to the user's smart card. This method is invoked on the client connected to the user's smart card. **Paramters:** *CVcertificate* – a card verifiable certificate to be processed by the smart card connected to the client **Actions:** the smart card verifies the certificate's signature and extracts the encapsulated public key for subsequent verification of the server's response (authentication token). **Returns:** true, if the certificate is valid, false otherwise.
String **getChallenge**() throws AuthenticationManagerException	Part of client authentication towards a server based on RSA. Generates a random challenge on the server to be sent to the client for signing. *This method is invoked on the server.* **Returns:** a 16-byte random *serverChallenge* (to be sent to the client)
String **getCertificate**(String *keyIdentifier*) throws CertificateManagerException	Part of client authentication towards a server based on RSA. Retrieves the certificate associated to the smart card's key pair for client authentication (referenced through keyIdentifier) to be sent to the server for validation *This method is invoked on the client connected to the user's smart card.*

	Parameters: *keyIdentifier* – the identifier of the certificate to be retrieved from the smart card (corresponding to the key pair associated to *keyIdentifier*) **Returns**: the certificate containing the public key from the key pair associated to *keyIdentifier*
boolean **verifyCertificate**(String *certificate*) throws CertificateManagerException	Part of client authentication towards a server based on RSA. This method is used to verify a certificate issued on a public key. *This method is invoked on the server.* **Parameters:** the *certificate* **Returns:** true, if the certificate is valid, false otherwise
String **clientAuth**(String *serverChallenge*, String *keyIdentifier*, String *PIN*) throws AuthenticationManagerException	Part of client authentication towards a server based on RSA. Client authentication is based on signing a challenge (and optionally additional information) retrieved from the server by the user's smart card connected to the client. This method is invoked on the client connected to the user's smart card. **Parameters:** *serverChallenge*: the 16 byte random challenge retrieved from the server *keyIdentifier*: the identifier associated to the private key to be used by the smart card to sign the requested authentication token *PIN*: the user's PIN to be checked on the smart card (authorizing the use of the card holder's private authentication key associated to *keyIdentifier*) **Returns:** The signed authentication token *cardAuthToken* retrieved from the smart card
Boolean **clientTokenVerify**(String *cardAuthToken*, String *keyAlias*) throws AuthenticationManagerException	Part of client authentication towards a server based on RSA. Client verification consists of verifying the authentication token retrieved from the client's smart card on the server. This method is invoked on the server. **Parameter:** *cardAuthToken*: the authentication token retrieved from the client's smart card and sent to the server *keyAlias*: the alias associated to the client user's public key to be used for verification on the server, and contained in the client certificate validated through the

	previous **verifyCertificate** command **[Note:** *this method can only be successfully invoked following a positive certificate validation on the server*.] **Returns:** true, if the client's authentication token has been verified, false otherwise
String **serverAuth**(String *cardChallenge*, String *keyAlias*, String *password*) throws AuthenticationManagerException	Part of server authentication towards a client based on RSA. Server authentication is based on signing a challenge (and optionally additional information) retrieved from the client's smart card by the server's private key. This method is invoked on the server. **Parameters:** *cardCchallenge*: the random challenge retrieved from the client's smart card *keyAlias*: the alias associated to the private key to be used by the server to sign the requested authentication token *password*: the password to access the key repository containing the server's private authentication key associated to *keyAlias* **Returns:** The signed authentication token *serverAuthToken*
Boolean **serverTokenVerify**(String *serverAuthToken*, String *keyAlias*) throws AuthenticationManagerException	Part of server authentication towards a client based on RSA. The authentication is performed by verifying the authentication token retrieved from the server on the client's smart card. This method is invoked on the client connected to the user's smart card. **Parameters:** *serverAuthToken*: the authentication token retrieved from the server to be verified by the client's smart card *keyAlias*: the alias associated to the server's public key to be used for verification on the client's smart card, and previously transmitted to the smart card through the **verifyCVCertificate** command **Returns:** true in case of successful validation of the signature, false otherwise **[Note:** *this method can only be successfully invoked following a positive certificate validation on-card and before card reset. In such case the extracted public key indicated by keyAlias is still held in the smart card's RAM.*]

11.5 GS (Group Signature)-Manager Interface

byte [] org.iso.mpeg.mxm.engine.securityengine.GSManager.**setup_parameters_GS** (String *algorithm*, String *GSAlias*) throws GSManagerException	This method generates the cryptographic parameters for setting up a group signature instance (in short terms: a group). In particular, it generates elliptic curve and pairing parameters, a group public key, a secret master key to be kept strictly with the group master; and a revocation list which is initially empty. This method must only be invoked by the group master, on an absolutely trustworthy platform. **Parameters:** *algorithm*: to be reserved for optional choices (like curve type, parameter size, etc.). In our initial release all specifying parameters will be hardcoded. *GSAlias*: the group alias to be used as a reference for the newly generated group instance. **Returns:** *GSparameters*, consisting of: *GSpublicparameters*: groups G_1, G_2, generator point g_1, group public key ω, hash-functions, pairing specification *GSsecretparameter*: secret master key γ (must strictly remain with the group master) Moreover: *MemberList*: list of secret keys of members, must remain secret to the group master; initially empty *RevocationList*: list of "revocation tokens", may be published to an authentication server; initially empty **Exceptions:** GSManagerException
byte [] org.iso.mpeg.mxm.engine.securityengine.GSManager.**generate_member_privatekey** (String *GSparameters*, String *IDuser*, *MemberList*) throws GSManagerException	This method generates a private key for a new group member to be accommodated to the group, enabling him or her to issue anonymous signatures on behalf of the group. The user's ID is only kept with the group master to govern its internal (secret) member list. This method must only be invoked on a trustworthy platform under the control of the group master. **[Note:** The group master can do two things with the computed private key: (a) it can embed it on a member's smart card, or (b) it can send it to the member via a trustworthy channel.**]** **Parameters:** *GSparameters*: the complete set of parameters defining the current group instance, as generated by

	"setup_parameters_GS" *IDuser*: an identifier for the user allowing maintenance of the member list *MemberList*: a reference to the member list to update with the new member's private key **Updates**: the member list; strictly confidential to the Group Master **Returns:** *SKmember*: the accommodated member's private key recipient's private key, enabling him or her to issue group signatures under the group instance defined by GSparameters. **Exceptions:** GSManagerException
byte [] org.iso.mpeg.mxm.engine.securityengine.GSManager.**revoke_member_privatekey** (String *GSparameters*, String *IDuser*, *MemberList*) throws GSManagerException	This method revokes a private key of an existing group member to be removed from the group, disabling him or her to furthermore issue anonymous signatures on behalf of the group. The user's ID is only kept with the group master to govern its internal (secret) member list. This method must only be invoked on a trustworthy platform under the control of the group master. **[Note:** Revocation works even against the revoked member's will, and even if that member refused to deliver his or her smart card.**]** **Parameters:** *GSparameters*: the complete set of parameters defining the current group instance, as generated by "setup_parameters_GS" *IDuser*: an identifier for the user allowing maintenance of the member list *MemberList*: a reference to the member list to updated with a revocation tag for the revoked member's private key. **Updates**: the member list; strictly confidential to the Group Master **Returns:** *RevocationToken*: the revoked member's revocation token, to be added to the revocation list. [**Note**: revocation tokens remain *anonymous*. Revoked member's IDs are *not* added in the revocation list, but stay with the group master.] **Exceptions:**

	GSManagerException
byte [] org.iso.mpeg.mxm.engine.securityengine.GSManager.**obtain_parameters_GS** (String *GSAlias*) throws GSManagerException	Provides the actual public parameters defining a group from a database. This method can be invoked by any outsider who wishes to verify a signature issued on behalf of the group referenced by *GSAlias.* **Parameters:** *GSAlias*: the alias associated to the specific group instance **Returns:** *GSpublicparameters*: groups G_1, G_2, generator point g_1, group public key ω, hash-functions, pairing specification
boolean org.iso.mpeg.mxm.engine.securityengine.GSManager.**perform_signaturecheck_GS** (byte[] *message*, byte [] *signature*, String *GSAlias*) throws GSManagerException	This method checks the validity of a group signature issued by a member of the group referenced through *GSAlias.* This method is invoked by any outsider wishing to verify a group signature. [**Note**: this method provides functionality only to check the cryptographic validity of the included signature. In order to complete the signature verification, a revocation check must be performed or requested.] **Parameters:** *message*: the message for which a signature is supplied. *signature*: an anonymous group signature issued on the message supplied. *GSAlias*: the alias associated to the group-instance under which the supplied signature has been issued. **Returns:** true if the signature has been found valid, false otherwise **Exceptions:** GSManagerException
boolean org.iso.mpeg.mxm.engine.securityengine.GSManager.**perform_revocationcheck_GS** (byte [] *signature* , String *RevocationList*, String *GSAlias*) throws GSManagerException	This method checks whether a group signature has been issued by a member who has been revoked from the group referenced through *GSAlias.* This method is invoked by any outsider wishing to verify a group signature, provided he disposes of a revocation list. [**Note**: this method provides functionality only to check whether the included signature has been issued by a group member whose token has been added to the revocation list. In order to complete signature verification, a (previous) signature check should have been performed.]

	Parameters: *signature* :an anonymous group signature issued on (any) message. *RevocationList* :list of "revocation tokens" (each anonymous token stands for one revoked member) *GSAlias* :the alias associated to the group-instance under which the supplied signature has been issued. **Returns:** *true* if the signature has been issued by non-revoked member valid, *false* if it has been issued by a revoked member. [**Note**: so "true" is the "positive" response, i.e. the signature has been issued by an active group member.] **Exceptions:** GSManagerException
boolean org.iso.mpeg.mxm.engine.securityengine.GSManager.**request_revocationcheck_GS** (byte [] *signature* , String *GSAlias*) throws GSManagerException	This method requests – for an included signature – to check whether the signature has been issued by a member who has been revoked from the group referenced by *GSAlias*. This method is invoked by any outsider wishing to verify a group signature, but having no access to a revocation list. The intended usage is to request the actual revocation check (not the revocation list itself!) from a trustworthy server. **[Note:** Unlike "perform_revocationcheck_GS" (where the revocation check is to be performed by the verifier himself), this is a *request* to a trusted *server* maintaining a revocation list, but not admitting access to it.] **Parameters:** *signature* – an anonymous group signature issued on (any) message. *GSAlias* - the alias associated to the group-instance under which the supplied signature has been issued. **Returns:** *true* if the signature has been issued by non-revoked member valid, *false* if it has been issued by a revoked member. [**Note**: so "true" is the "positive" response, i.e. the signature has been issued by an active group member.] **Exceptions:** GSManagerException
byte [] org.iso.mpeg.mxm.engine.securityen	This method is used to sign a message under the group instance referenced by GS*Alias* on a smart card. The

gine.GSManager.**sign_message_GS_SC** (byte[] *message*, String *GSAlias*, String *PIN*) throws GSManagerException	corresponding system parameters and the member's private group signature key are assumed to be stored on the card and addressed by GS*Alias*. Note that the private key signing operation is performed on-card and therefore the private signature key will never leave the secure smart card. This method is invoked by group members anonymously signing messages on behalf of the group. **Parameters:** *message*: the message for which a signature is required. *GSAlias*: the alias associated to the group-instance under which the required signature shall be issued. *PIN*: the password to authorize the signature operation on the member's smart card **Returns:** *signature*: the anonymous group signature issued on the supplied message. **Exceptions:** GSManagerException
byte [] org.iso.mpeg.mxm.engine.securityengine.GSManager.**sign_message_GS** (byte[] *message*, String *GSAlias*, String *SKmember*, String *password*) throws GSManagerException	This method is used to sign a message under the group instance referenced by GS*Alias*. This method is invoked by group members anonymously signing messages on behalf of the group. **Parameters:** *message*: the message for which a signature is required. *GSAlias*: the alias associated to the group-instance under which the required signature shall be issued. *SKmember*: the private group signature key of the group member; or empty string *password*: the password to access the key repository **Synopsis:** the user shall either enter his private group signature key *SKmember* directly, or leave *SKmember* as an empty string and instead request the private group signature key from a safe repository and enter the corresponding *password*. **[Note:** the key repository will usually be a software repository in this case. In case no safe repository is used, the parameter *password* is obsolete.] **Returns:** *signature*: the anonymous group signature issued on the supplied message.

	Exceptions: GSManagerException