

Project IST-FP6-026476 SEAMLESS
“Small Enterprises Accessing the Electronic Market of the Enlarged Europe by a Smart Service Infrastructure”
STREP – Information Society Technologies (IST)

Deliverable D3.2.2
Trust and Security
Technical Specification

Workpackage WP3 – Technological Infrastructure
Task T3.2 – Trust and security functionality

Abstract

This document presents how the security issues have been addressed in the development of the SEAMLESS framework, having into account the presentation of requirements and constraints, as well as the technological choices that have been analysed initially in the overall platform architecture. As part of the research work done, the structure of components of the SEAMLESS framework has been analysed (including the Registry and Repository infrastructure adopted from the SEEMseed project) in order to provide consistent security management (authentication / authorisation, data/message integrity, confidentiality, transaction integrity, privacy) across the whole platform.

The selected solution includes XML Digital Signatures to sign XML documents and binary data files, in conjunction with guidelines to use SSL connections within the SEAMLESS network. The complete solution is described in terms of software components that have been developed and/or selected based on open source developments and that have been appropriately combined and fine tuned to provide a complete and stable set of security services towards the applications that will access the SEAMLESS framework.

This report includes a description of the interfaces to be used in the SEAMLESS platform components implementation.

Start date of project	Jan 1 st , 2006	Duration of project	30 months
Deliverable due date	Jan 21 st , 2007	Actual submission date	Jan 30 th , 2007
Dissemination level	PU	Revision status	Final
Responsible partner	ATC ROM	Authors	T3.2 participants

Change record

Rev. N.	Description	Author	Date	Review
0	Full draft	D. Baltas (ATC ROM) L. Moga (ATC ROM) S. Turbut (ATC ROM)	Dec 11, 2006	S. Campbell (TIE) F. Bonfatti (U MODENA)
1	Final version	D. Baltas (ATC ROM) S. Turbut (ATC ROM)	Jan 17, 2007	J.V. Vidagany (ANTARA)

Table of contents

ABBREVIATIONS	4
1 EXECUTIVE SUMMARY	5
2 SECURITY DESIGN OVERVIEW	6
2.1 SEAMLESS Architecture Overview	6
2.2 Security requirements	7
2.2.1 General requirements on policy	8
2.2.2 Requirements on accountability	8
2.2.3 Requirements on assurance	9
2.2.4 Specific requirements on SEAMLESS security functionality	9
2.2.5 Requirements on implementation	9
2.3 Security functionalities	10
2.4 Networking Considerations	11
2.4.1 Communication types	11
3 SECURITY AND TRUST TECHNICAL APPROACH	12
3.1 Technology selection overview	12
3.1.1 Encryption	12
3.1.2 The SAML standard (OASIS)	13
3.1.3 The XKMS standard (W3C)	15
3.1.4 Other XML-based standards (W3C)	17
3.1.5 SSL	18
3.2 SEAMLESS security platform functional approach	19
3.2.1 Transport level	21
3.2.2 Application level: authentication	21
3.2.3 Application level: authorisation	22
3.2.4 Application level: data integrity and non-repudiation	24
3.2.5 Application level: data confidentiality	24
3.3 Pilot implementation approach	27
3.3.1 Infrastructure	27
3.3.2 Software modules: AAInterface	29
3.3.3 Software modules: SecureIT	34
3.3.4 Software modules: Logger	35
4 INTERACTION EXAMPLE WITH THE SRRN	38
4.1 From signature & encryption point of view	38
4.2 From authentication & authorization point of view	39
5 CONCLUSIONS	41
ANNEX I – SEEMSEED (SRRN) SECURITY SUPPORT	42
ANNEX II – CLASS DIAGRAM, SECURITY MODULE	44
ANNEX III – SECURITY INTERACTIONS SEQUENCE DIAGRAMS	45
REFERENCES	47

Abbreviations

Acronyms used in present document are listed below:

HTTP	-	Hypertext Transfer Protocol
HTTPS	-	Hypertext Transfer Protocol over SSL
PKI	-	Public Key Infrastructure
SAML	-	Security Assertion Markup Language
SOAP	-	Simple Object Access Protocol
SQL	-	Structured Query Language
SSL	-	Secure Socket Layer
TLS	-	Transport Layer Security
URL	-	Uniform Resource Locator
XML	-	Extensible Markup Language
SRRN	-	SEAMLESS Repository - Registry Network

1 Executive Summary

In this document it is presented how the security and trust issues have been addressed in the SEAMLESS framework development. It is organized in four chapters:

The first chapter goes over the security considerations presented in the overall architecture of the platform. It describes the requirements and constraints that guide the definition of the appropriate security approach to adopt. The security functionalities namely authentication and authorization, data and message integrity, user confidentiality, transaction integrity and privacy are introduced.

In the second chapter we present an overview of the technologies that are used, namely public and private key encryption, the use of digital signatures and the SAML standard and SSL for the establishment of secure connections. We provide justification of the respective selection and discussed the detailed way public and private key encryption/decryption and digital signatures will be used to secure data exchange over the distributed SEAMLESS network. Then, the possibility to have a distributed network of identity providers, to be managed by the mediator (optional) is justified, and the components that need to be installed at each mediator is enumerated. Finally the software modules that have been developed and the way they must be used within the framework in order to maintain the necessary level of security is presented.

Based on the fact that the SEEMseed SRRN prototype has been selected to support the SEAMLESS implementation, the third chapter is devoted on presenting examples that show how the security implementation supports the implementation of the register and repository network.

The document closes with a review of the security support within the SRRN and a presentation of the class and sequence diagrams for the security modules.



2 Security Design Overview

The purpose of this chapter is to define the mechanisms which will make SEAMLESS a trusted, secure and reliable system. The functionalities which will be described will minimize the potential SEAMLESS exposure to unauthorized access and use of system resources.

2.1 SEAMLESS Architecture Overview

The figure below depicts the SEAMLESS general architecture. Based on this, and following a presentation of the main security requirements identified during the design of the system, the way of securing the communication between system components and external users (or applications) is analysed. At the end of this chapter the security modules that will be built and the relations between them are presented.

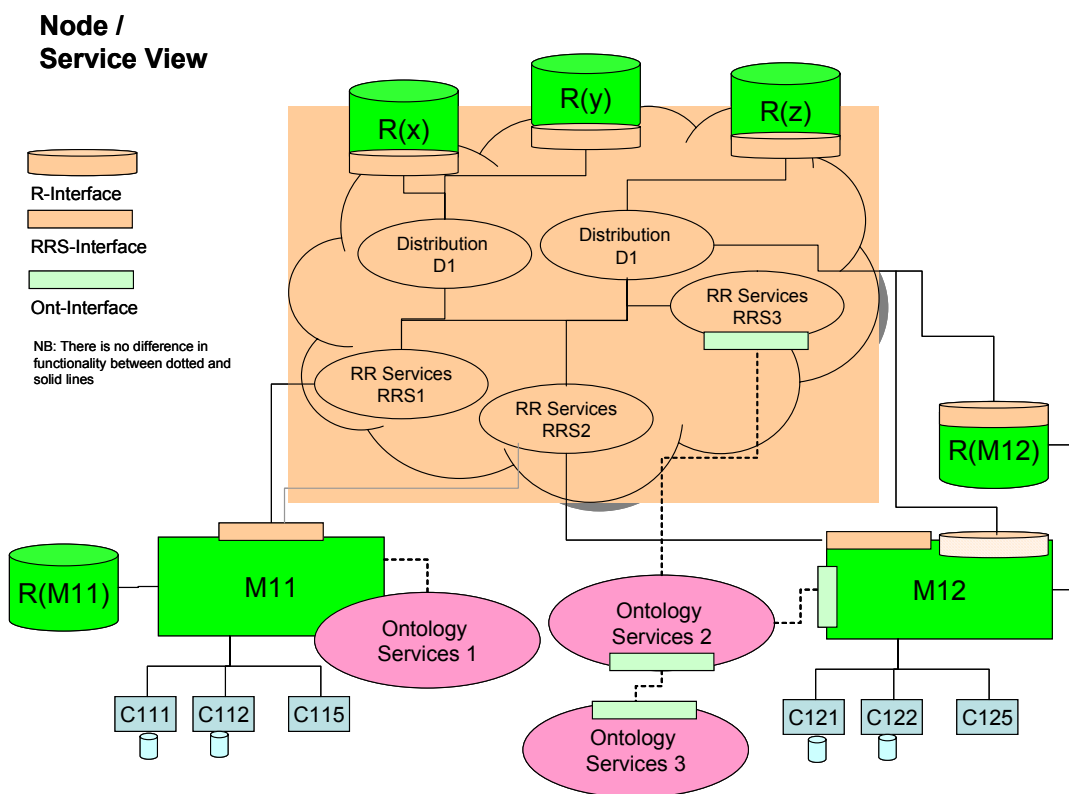


Figure 1 SEAMLESS General Architecture

In the SEAMLESS general architecture we can identify three functional areas, together with the associated services:

End User Area, that includes the services offered by the SEAMLESS platform to an end user (the end user in SEAMLESS can be a human user or a system running on the company side, e.g. an ERP system).

Mediator Area, provides mediator services like searching or sending documents, and the semantic translation of this information.

SRRN Area, provides the Semantic Service and the Storage Service.

Introducing an initial design consideration that will be detailed in the following paragraphs, the SEAMLES Security Platform Module(s) will come to support the SEAMLESS architecture as a

horizontal set of services which will serve the areas presented above, as depicted in the following diagram.

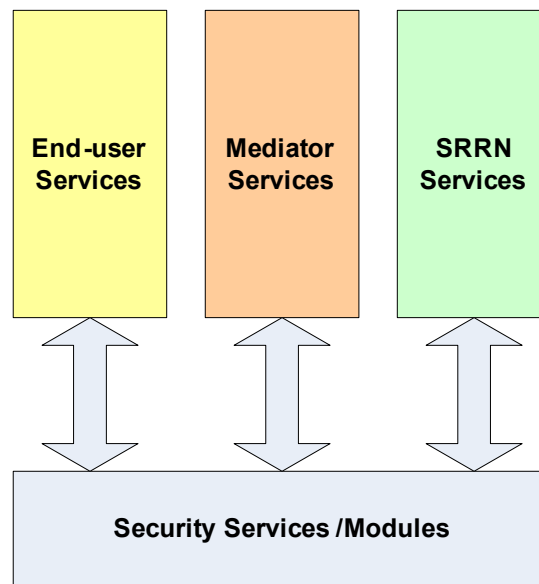


Figure 2 Security Service – Horizontal Service

In the next sections it will be analyzed how these services will be made available either as specific web services or as software components to be embedded into the software modules that implement the different levels of services.

2.2 Security requirements

The requirements regarding security in the proposed SEAMLESS platform are broadly outlined in the project technical annex and have been depicted in more detail also from the overall architecture proposal (Deliverable 3.1). The security module(s) and platform components are intended to ensure security and trust inside the SEAMLESS framework.

In general, secure systems control the access to information, so that only properly authorized parties or processes operating on their behalf, have access to read, modify, create or delete information.

One of the main characteristic of the Registry and Repository system (SRRN) architecture is its distributed nature. Based on this premise and the discussion and architecture presentation above, the following broad requirements can be identified immediately:

- Trust management should be distributed (the security framework should be able to manage the necessary credentials presented by a user and decide whether to trust it or not, based on local policies).
- Specification and enforcement of security policy should be separate from the applications, and the framework needs to be designed and implemented so as to allow policies to evolve dynamically without modifying the application to which these policies apply, in support of the flexibility and scalability of the whole SEAMLESS framework.

Starting from the SRRN core, the infrastructure is designed to be intrinsically secure, so as to assure the necessary degree of trust to the collaboration environment. This means introducing user authentication and authorisation at the application level, digital signature and verification functions to sign documents using a certificate, layer server authentication for SSL communication at all the levels of the architecture.

This creates the need of trusted certification authorities *within the SEAMLESS framework* that will support the above requirements.

Figure 3 below, represents the three levels identified in the system overall architecture in an alternative conceptual representation, taking into consideration the distributed ontology components that comprise the core of the SEAMLESS paradigm. In order to make the correspondence with the previous functional representation, we note that physically the second and the third level will be local - most probably in the mediator(s) (per country or per sector). From the certification authority point of view, regardless of the standards that will be used, we need to have:

- LRA: Local Registration Authority, which will cover the second and the third level.
- MRA: Master Registration Authority, which will cover the first level.

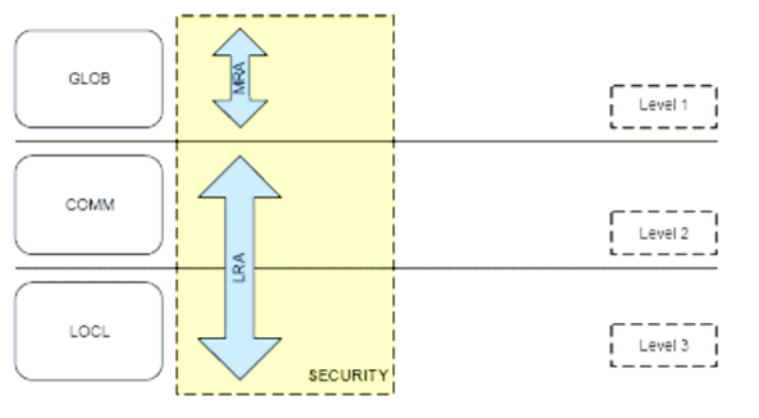


Figure 3: Certification authorities in SEAMLESS hierarchy

MRA will be in contact with the LRAs and this will be transparent for the nodes existing on the second and third level. This way any changes performed in an LRA at some point can be observed by other LRA residing in another country for example.

Based on best practices recommendations and state-of-the-art capabilities the following is an enumeration of broad requirements that the SEAMLESS overall architecture, and the security framework design and implementation has in mind and has pursued to address convincingly and effectively (adapted for completeness from the Overall Platform Architecture document [1]).

2.2.1 General requirements on policy

R1 – SECURITY. There must be an explicit and well-defined security policy enforced by the system, i.e. a set of rules that are used by the system to determine whether a given subject can be permitted to gain access to a specific resource.

Computer systems of interest must enforce a mandatory security policy that can effectively implement access rules for handling sensitive (e.g. confidential) information. These rules include requirements such as: No person lacking proper clearance shall obtain access to classified information. In addition, role based security controls primarily are required to ensure that only selected users or groups of users may obtain access to data (e.g., based on a need-to-know).

R2 – MARKING. Access control classification must be associated with objects, in order to control access to information stored in a computer, according to the rules of a mandatory security policy. This is also typically extended to the modes of access accorded to those who may potentially access the object.

2.2.2 Requirements on accountability

R3 – IDENTIFICATION. Individual subjects must be identified

Each access to information must be filtered based on who is accessing the information and what classes of information they are authorized to deal with. This identification and authorization information must be securely maintained by the computer system and be associated with every active element that performs some security-relevant action in the system.

R4 – ACCOUNTABILITY. Audit information must be selectively kept and protected so that actions affecting security can be traced to the responsible party.

A trusted system must be able to record the occurrences of security-relevant events in an audit log. The capability to select the audit events to be recorded is necessary to minimize the expense of auditing and to allow efficient analysis. Audit data must be protected from modification and unauthorized destruction to permit detection and after-the-fact investigations of security violations.

2.2.3 Requirements on assurance

R5 – ASSURANCE. The computer system must contain hardware/software mechanisms that can be independently evaluated to provide sufficient assurance that the system enforces requirements 1 through 4 above.

In order to assure that the four requirements of Security Policy, Marking, Identification, and Accountability are enforced by a computer system, there must be some identified and unified collection of hardware and software controls that perform those functions. These mechanisms are typically embedded in the operating system and are designed to carry out the assigned tasks in a secure manner. The basis for trusting such system mechanisms in their operational setting must be clearly documented such that it is possible to independently examine the evidence to evaluate their sufficiency.

R6 – CONTINUOUS PROTECTION. The trusted mechanisms that enforce these basic requirements must be continuously protected against tampering and/or unauthorized changes. No computer system can be considered truly secure if the basic hardware and software mechanisms that enforce the security policy are themselves subject to unauthorized modification or subversion. The continuous protection requirement has direct implications throughout the computer system's life-cycle.

2.2.4 Specific requirements on SEAMLESS security functionality

RS6 – SECURITY COVERAGE – The security module(s) should be able to provide unequivocally the functionality to be used in order to support secure exchange of data between nodes no matter where they are physically located.

At (each if more than one) SRRN layer the user id and his role into the SEAMLESS system should be known.

RS8 MEDIATOR SECURITY ROBUSTNESS - The security module should ensure that in case a node (or more) are down the authentication process should still be in place and maintain the security of the system by administering appropriate access control ONLY to users that can be appropriately identified.

RS9 TRANSPARENCY – SSO - SEAMLESS security should be transparent to the end-user. Once authenticated, the user should not provide credentials to another SRRN node where his request may be issued. The last requirement is called in the literature SSO (Single Sign On). End users and equally operators tend in general to walk away from complex solutions (and this is especially true for SMEs that do not have internally extensive IT expertise in most cases). The platform has therefore to be capable to hide the complexities of maintaining sufficient level of security behind the transactions supported.

2.2.5 Requirements on implementation

R10 – OPEN SOURCE COMPLIANCE – The implementation should make use of open source software and apply the principles and licensing rules of the OSC.

R11 – IMPLEMENTATION ECONOMY – The implementation of the security functionality should try to avoid overheads in the sense of refactoring existing implementation components (referring mainly to what is available from the SEEMseed project)



2.3 Security functionalities

Security and trust in the support and practical deployment of web services is a high priority issue especially nowadays when the web service paradigm is proliferating given its conceptual as well as practical appeal. The technical community is therefore working elaborating a set of standard functions that any system can implement in order to ensure trust and security.

The main functionalities which will be invoked to address the above requirements and then cover the security issues are listed below.

Authentication. Is the function that concerns the retrieval of the identity of a user or process from a request. Authentication plays a key role in forming the basis for enforcing access control rules (the authentication part).

Authorisation. It handles access control typically role based. At the application layer the SEAMLESS system has identified and needs to support multiple roles (at the user, at the mediator and at the SRRN levels).

A sample of the roles/permissions that can be applied are the following:

At the Company:

- *user*, can access the user only section of the SEAMLESS application.
- *superuser*, same permissions as *user* role plus the permission to manage the corresponding registered company profile.

At the Mediator

- *administrator*, this role is generic describing the operator(s) acting in the name of the mediator.

The functionality Authentication is shared as part of the user management module with the applications at the mediator. User management is considered in our approach as separate from the security framework since it is more pertinent to operation, and can be provided based on additional technologies like user certificates, etc...

Nevertheless is it necessary that the security implementation will interact appropriately with the user management application module(s) at the mediator to retrieve the information necessary for its own operations.

Non-repudiation. This is the functionality that ensures that a message has indeed been sent and received by the parties claiming to have sent and received the message. This functionality provides the proof that the receiver cannot deny having received the message. Non-repudiation is an important requirement to establish trust especially in the execution of commercially binding transactions. It can in general be accomplished through:

- **Digital signatures.** A digital signature is used as a unique identifier for an individual, much like a written signature and can be used as evidence that the sender really did create the data. After attaching a digital signature to a document the sender can no longer reasonably deny not to have signed the data. In this respect XML Digital Signatures provide non-repudiation of origin and they are considered as the major tool within the SEAMLESS security framework.
- **Confirmation services.** Where the message transfer agent can create digital receipts to indicate that messages were sent and/or received.
- **Timestamps.** Timestamps contain the date and time a document was composed and proves that a document existed at a certain time,

with digital signature being the technical approach that is selected within SEAMLESS.

Data integrity. Data integrity means that a message received is identical to the message that was sent, guaranteeing that data is not changed in transit, whether deliberately or by accident.

Privacy and confidentiality. Electronic records, files and communication should be protected from unauthorized access.



Confidentiality of the data in transit is achieved by the use of the Secure Sockets Layer (SSL) protocol when establishing the communication path between two nodes (as further elaborated below). SSL provides transient confidentiality of a message between two nodes.

Data integrity, privacy and confidentiality are mainly achieved with the use of an encryption and decryption scheme (symmetric or asymmetric) and this is the approach that will be also adopted as the major paradigm in the security platform for SEAMLESS.

2.4 Networking Considerations

Given the distributed p2p architecture adopted in SRRN, networking and information flow issues are taken into consideration during the process of designing the security of the system.

2.4.1 Communication types

In this direction, we note that the mediator communication with the end user is taking place outside the SRRN system (which relies on the initial implementation of SEEMseed). This environment between the user and the mediator has to be specially considered.

On the other hand the mediator will communicate with the SRRN through the respective RR Services and an appropriate security mechanism is already in place within the SRRN core (as part of the initial SEEMseed platform - described in Annex 1) and will be used to support essentially 'internal' SEAMLESS information exchange in conjunction with the mechanisms that will be additionally implemented as part of the SEAMLESS security framework to have a complete and compatible security solution.

Regardless, based on the general architecture of the SEAMLESS platform (cf. Figure 1) we have identified the following communication paths that will have to be secured in the operation of the SEAMLESS system:

- **User-to-Application.** This type of communication appears for example when the end-user is issuing queries to the SEAMLESS application. These queries can be issued through the browser or if issued automatically, through web services.
- **Application-to-SEAMLESS (mediator and semantic) services.** The front end applications at the mediators will process the users' queries and it will take the necessary actions like storing a document or sending a query along the SRRN system (which are implemented as SEAMLESS services), and translating it properly to be understood at the end point.
- **SEAMLESS (mediator and semantic) services-to-SRRN.** The SEAMLESS services will query the correspondent SRRN services in order to fulfil the request issued by the application.
- **SRRN-to-SRRN.** Although in the initial approach Due to the specific architecture (distributed) of the SRRN system, the queries will be passed to various nodes of the SRRN network.
- **SRRN-to-external systems.** If one SRRN registry is connected to an external registry (UDDI), the respective node will query also the external one.

Note: From the perspective of security mechanisms within the SEAMLESS security platform we readily identify an issue, that the trust and security when communicating with external registries (like UDDI for example). In such an event it is only possible to use SSL technology (presented below) in order to ensure a level of security between two nodes. In addition to this it is necessary according to the approach we discuss below in order to secure information exchange to install or at least configure specific software at both sides. This can be done only if custom software can be installed on the remote side (in which case the API of the SEAMLESS security toolkit can be used). If this not the case however security can be ensured only at the communication level, through SSL.

3 Security and trust technical approach

This section will present the approach followed to meet the requirements described in the previous section. Based on the functional approach, the implementations details and the needed infrastructure will be also described. The overall proposal concerning implementation of the technical platform for the delivery of security and trust services within SEAMLESS is to use mechanisms invoking the following technical means:

- Use SSL encryption and authentication for servers that expose SEAMLESS services.
- Use XML Digital Signatures (DSig) to authenticate users accessing services.
- Use public and private key encryption to securely identify data and maintain them confidential

The approach that has been discussed above in general terms and agreed in SEAMLESS is in line with the current strategies in securing XML transactions in e-business.

Through the means above it is possible to support effectively:

Non-Repudiation of Origin. Using digital signatures can be used by the recipient of a document (through a SEAMLESS call) as evidence that the sender really did create the data. After attaching a digital signature to a document the sender can not deny not to have signed the data.

Integrity. Digital signatures and key based encryption can also be used to ensure that:

- Data has not been altered since it was signed and published
- Ownership of a particular data segment can be validated
- Confidence that data transferred among SEAMLESS nodes can be assured

Confidentiality: Confidentiality of the data in transit is achieved by the Secure Sockets Layer (SSL) protocol. SSL provides transient confidentiality of a message between two nodes.

User Authorization: When a certificate is used to authenticate the user, the system can define, store, and then extract information from that certificate to correctly identify the user, verify access rights and role and to proceed with functions available from the SEAMLESS infrastructure.

These mechanisms have been available in an early prototype at the elements of SRRN (cf. Annex 1). However it has been necessary to build a toolkit with security components for support in the additional components that appear as part of the SEAMLESS architecture extending the authentication features and adding access control to resources, by using mechanism like SAML as will be explained in detail in the following sections.

3.1 Technology selection overview

The following overview is provided for completeness and has been included and extended from the SEAMLESS Overall Architecture Design report[1]

3.1.1 Encryption

Encryption is any procedure used to convert plain text into cipher text in order to prevent anyone except the intended recipient from reading that data. There are many types of data encryption, and they are the basis of network security.

There are two basic techniques: symmetric encryption (also called public key encryption) and asymmetric encryption (private-public key encryption.)

Symmetric encryption

Symmetric encryption is the oldest and best-known technique. A secret key, which can be a number, a word, or just a string of random letters, is applied to the text of a message to change the content in a particular way. The process is reversible and as long as both sender and recipient know the secret key, they can encrypt and decrypt all messages that use this key.



Symmetric ciphers are significantly faster than asymmetric ciphers, but the requirements for key exchange make them difficult to use.

Asymmetric encryption

The problem with secret keys is exchanging them over the Internet or a large network while preventing them from falling into the wrong hands. Anyone who knows the secret key can decrypt the message. One answer is asymmetric encryption, in which there are two related keys--a key pair. A public key is made freely available to anyone who might want to send you a message. A second, private key is kept secret, so that only the initiator person or application knows it. Any message (text, binary files, or documents) that are encrypted by using the public key can only be decrypted by applying the same algorithm, but by using the matching private key. Any message that is encrypted by using the private key can only be decrypted by using the matching public key.

This means that one do not have to worry about passing public keys over the Internet (the keys are supposed to be public).

A problem with asymmetric encryption, however, is that in implementation it is slower than symmetric encryption. It requires far more processing power to both encrypt and decrypt the content of the message.

3.1.2 The SAML standard (OASIS)

SAML stands for "The Security Assertion Markup Language" and it was designed to protect the privacy and confidentiality of identity information. The single most important problem that SAML is trying to solve is the web single sign-on (SSO) problem. SAML has become the definitive standard underlying many web SSO solutions in the identity management problem space. SAML standardizes the full range of functions associated with receiving, transmitting, and sharing security information to:

- Provide XML formats for user security information and formats to request and transmit the information.
- Define how these messages work with protocols such as SOAP.
- Specify precise message exchanges for certain common use cases, such as Web SSO.
- Support a number of privacy protection mechanisms, including the ability to determine users' attributes without revealing their identities.
- Detail how to handle identity information in formats provided by widely used technologies, including Unix, Microsoft Windows, X.509, and LDAP, DCE, and XCMML.
- Formulate a metadata schema that allows participating systems to communicate the SAML options they support.

SAML Roles, Assertions, and Statements

A federated environment involves at least three roles.

- **Relying Party** - makes use of the identity information (typically this is a SEAMLESS Service Provider that decides what requests to allow)
- **Asserting Party** - provides the security information; SAML calls this the "**Identity Provider**"
- **Subject** - the user associated with the Identity Information

In SEAMLESS, there will be many subjects and several Service Providers. There also will be multiple Identity Providers. Basically, the Service Provider or the Relying Party needs to know three things:

- The identity information
- That the party making a request is the Subject
- That the Identity Provider is trusted to provide this information

In SAML, an **Assertion** carries the information. The Assertion contains header information, the name of the Subject, and one or more **Statements**. The header contains the name of the Identity Provider and other information such as issue and expiration dates.



The two most important Statement types are:

- **Authentication Statements** - report that the Subject was authenticated using a particular method at a specific time and place (SAML defines the details of more than 20 different authentication methods. Authentication statements support SSO, where the Identity Provider performs the login on behalf of the Service Provider).
- **Attribute Statements** - contain properties associated with the Subject. Groups and Roles are typical attributes, but financial data or any other property could be carried in an Attribute Statement.

An Assertion can carry both types of Statements.

Bindings and Profiles

Although SAML Assertions travel from an Identity Provider to a Service Provider, they can do so through a number of possible paths as an assessment of the SAML flexibility. In a Web services environment, the SOAP header can carry them. SAML defines a set of request and response messages in XML that can be used by a Service Provider to obtain Assertions directly. The requests specify what the Service Provider wants—for example, "all the attributes of user John Smith". The responses return one or more Assertions that match the request. To allow different products to interoperate, it is also necessary to specify how the network protocols will carry the requests and responses.

The SAML SOAP Binding specifies how to carry them in the body of a SOAP message.

SEAMLESS is a distributed environment and in order to achieve authorization it is necessary to remove dependence on localized user management. This is where SAML comes in, enabling authentication and authorization information to be shared by multiple parties across security domains. The SEAMLESS architecture is based on the use of Web Services. A basic principle of security is the need to secure all the levels involved in the Web Service, any weak level will permit attackers to penetrate the system. These levels include:

- Workflow or business process level
- Cataloguing and description of Web Services
- Communications level (typically SOAP)
- Storage of XML documents

Web services levels

If a Web Service moves to another security domain, it must carry its security restrictions so that the other domain can properly handle the Web Service. As already state, this propagation can be performed through SAML.

Communications level

The Simple Object Access Protocol (SOAP) defines the communications level. There are other protocols proposed for this level, but they have not gained general acceptance. SOAP itself has no security; all of its security comes from the SOAP Security Extensions. The communication can be encrypted using the W3C XML Encryption protocol and the XKMS standard (please see bellow). Super-encryption is possible, where the whole message with parts encrypted can be encrypted again, as is done when SSL is used for secure transmission over HTTP.

A SOAP message includes a header and a payload. As shown in the following diagram, SAML assertions can be included in the header or in the payload (see Figure 4).



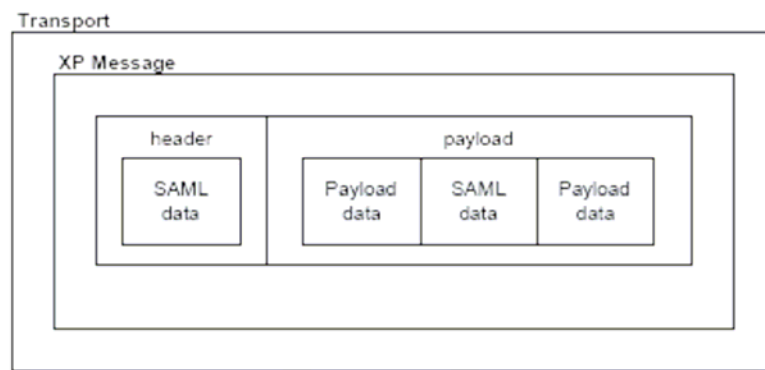


Figure 4 SOAP Wrapper with SAML header and payload schematic

The XML encryption protects the secrecy of the message, while a Public Key Infrastructure (PKI) can be used to provide digital signatures, and key distribution. This PKI can be simplified by the use of XML Key Management Specification (XKMS), intended for the integration of PKI and digital certificates. For example, digital signature processing can be delegated to a Web Service in order to further simplify the PKI structure.

To summarize SAML will ensure to SEAMLESS:

- Authentication from different domains. Enables global single sign-on and access control.
- Authentication mechanism neutrality. Permits seamless integration of future authentication mechanisms.
- Centralized credential management. Single point of control for managing credentials, thus, provides easy administration.

The full benefit of SAML regarding the SEAMLESS system is realized where parties with no direct knowledge of each other will interact via a third-party introduction (where the authorisation will be performed).

3.1.3 The XKMS standard (W3C)

XKMS stands for “XML Key Management Specification” and it is a protocol distributing and registering public keys. It is an open standard to simplify securing the XML-based transactions using PKI (Public Key Infrastructure) and digital certificates. Securing the exchange of XML documents with the use of the XKMS is considered best practice.

XKMS allows easy management of the PKI by abstracting the complexity of managing the PKI from client applications to a trusted third party. The trusted third party hosts the XKMS service while providing a PKI interface to the client applications. XKMS is offering an XML interface to PKI, thus, reducing the implementation effort and in the same time offering the benefits of PKI.

XKMS objectives

The primary objectives of XKMS are:

- Create an abstract layer between the application and the PKI solution. This allows the application to plug in different PKI solutions based on the need, without requiring any modification of the application itself.
- Eliminate the need for the application to understand complex PKI syntax and semantics by providing a simple XML-based protocol for processing key information through the XKMS service.
- Move complexity from the client application to the infrastructure level, thereby allowing the application to remain simpler and smaller. This allows even small footprint devices to take advantage of PKI.

- Implement XKMS such that it is platform-, vendor-, and transport protocol-neutral.

XKMS overview

XKMS is implemented as a Web service that allows a client application to access PKI features, thereby reducing the client application's complexity. The client application need not be concerned about the syntax of the underlying PKI, which could be any of the following:

- X.509 (the most widely used)
- Simple Public Key Infrastructure (SPKI)
- Public Key Infrastructure X.509 (PKIX)

The XML Key Management Specification allows for easy management of the security infrastructure, while the Security Assertion Markup Language makes trust portable.

XKMS stands for the XML Key Management Specification and consists of two parts:

- **XKISS** (XML Key Information Service Specification) and
- **XKRSS** (XML Key Registration Service Specification).

XKISS defines a protocol for resolving or validating public keys contained in signed and encrypted XML documents, while XKRSS defines a protocol for public key registration, revocation, and recovery. The key aspect of XKMS is that it serves as a protocol specification between an XKMS client and an XKMS server in which the XKMS server provides trust services to its clients by performing various PKI (public key infrastructure) operations, such as public key validation, registration, recovery, and revocation on behalf of the clients.

XKISS - this part of XKMS addresses the mechanism that allows client applications to authenticate encrypted/signed data.

The client authenticates the encrypted/signed data by passing the corresponding key information to the service provider. The service provider then responds with "true" or "false." "True" indicates that the public key corresponding to the private key used for signing does belong to the entity that claims to have signed the docs. Otherwise "false".

The XKISS service specification defines the following two operations:

- **Locate:** Locate resolves a `<ds:KeyInfo>` element that may be associated with XML encryption or XML signature, but it does not prove the validity of data binding in the `<ds:KeyInfo>` element.
- **Validate:** This operation does all that locate does, plus more. The locate service finds a key based on the `<ds:KeyInfo>` element, but does not assure the trustworthiness of the key binding information. The validate operation not only searches the public key corresponding to the `<ds:KeyInfo>` element, but also assures that the key binding information that it returns is trustworthy.

An XKRSS service specification defines four operations:

- **Register:** Information is bound to a key pair through key binding. During registration, either the client provides the public key, along with some proof of possession of the corresponding private key, or the service provider generates the key pair for the client. The service provider may request more information from the client before it registers the public key (and optionally the private key as well).
- **Reissue:** A previously registered key binding is reissued. New credentials in the underlying PKI are generated using this operation. While there is no lifespan for the key binding information used by XKMS, the credentials issued by the underlying PKI occasionally do have a time span that must be renewed periodically.

- **Revoke:** This operation allows clients to destroy the data objects to which a key is bound. For example, an X.509 certificate that's bound to an XKMS key is destroyed when this operation is called.
- **Recover:** This operation allows clients to recover the private key. For this operation to be meaningful, the private key must have been registered with the service provider. One of the ways in which the service provider may have the private key is when the key pair is generated at the server rather than the client.

The recover operation adds a lot of value in cases where the private key of the pair is used to encrypt data; if the private key were lost, the encrypted data would also be lost. When this happens, it makes sense to have the private key generated at the server and have the service provider keep the private key. That way the private key can be recovered.

If the private key that's lost is only used for digitally signing the documents, a new key may be generated without affecting the validity of the existing signed documents. Hence, keys used for such a purpose can very safely be generated by the client, and the private key may never be registered with the XKMS service.

An XKMS service implementing XKRSS service specifications may choose to offer some, all, or none of these operations. The XKRSS service specification does not make it mandatory for the XKMS service to implement any of the operations.

By using XKMS we can ensure the digital signatures, encryption services and certificate processing inside the SEAMLESS system. There are several implementations for key/certificate registry. When you need to retrieve an external key of certificate you may want to use XKMS. XKMS is specialized for PKI. CAs such as Verisign provide key management services with XKMS. XKMS suggests that key management can be outsourced to an independent third party. It could even be a Web Service. In the context of SEAMLESS the Certification Authority can even be internal to the system situated at the mediator or a regional/national public administration.

PKI (Public Key Infrastructure)

PKI is making use of a set of public and private keys. The private keys will be held by individuals, while the public keys can be distributed widely. The parties involved in an XML exchange can securely communicate with each other without prior business arrangement, this being a business requirement generated by the project nature. Indeed, it is possible at some point that two nodes of the system communicate without knowing in advance the corresponding keys. SEAMLESS will implement a Trust Finder service (will implement a Locate and Validate services), by which an organization or process can obtain the key to be used in the transaction and also the receiver can be obtained from the signature the identity of the sender.

XKMS by its nature will have the possibility to deal with the situation when the two parties which are exchanging XML documents are using different PKI protocols, which again proves useful in the SEAMLESS context where this situation is more than likely to happen.

XKMS has also the advantage that it can be implemented client-to-client, server-to-server, server-to-client and so forth. Inside the SEAMLESS architecture the XKMS services will be represented as Web Services. The main use of XKMS in SEAMLESS is to secure the SAML traffic.

3.1.4 Other XML-based standards (W3C)

XML Digital Signature

XML signatures are digital signatures designed for use in XML transactions. The standard defines a schema for capturing the result of a digital signature operation applied to arbitrary (but often XML) data.

A fundamental feature of XML Signature is the ability to sign only specific portions of the XML tree rather than the complete document. XML Digital Signature will be useful in the SEAMLESS context as this specification is used in the SAML specification.

XML Canonicalisation



Any XML document is part of a set of XML documents that are logically equivalent within an application context, but which vary in physical representation based on syntactic changes permitted by XML 1.0 and Namespaces in XML. This specification describes a method for generating a physical representation, the canonical form, of an XML document that accounts for the permissible changes. Except for limitations regarding a few unusual cases, if two documents have the same canonical form, then the two documents are logically equivalent within the given application context. The application of this specification in the context of SEAMLESS project is the following. The signing authority will digest and sign the canonical form of the document instead of the original form. Similarly, at the time of verification, the canonical form will be verified instead of the original XML. Thus all logically equivalent versions of the signed XML document will result in successful verification (validation) of XML digital signatures.

XML Encryption

Using this specification one can encrypt a whole XML document, a single element or the content of one element. In the same time the specifications can be used to encrypt also non-XML data like a JPEG file for example. The SEAMLESS system will use this specification to ensure the secrecy of the XML document exchange.

3.1.5 SSL

The Secure Socket Layer (SSL) along with the de facto Transport Layer Security (TLS) is used to provide transport level security for web services applications. SSL/TLS offers several security features including authentication, data integrity and data confidentiality. SSL/TLS enables point-to-point secure sessions. It doesn't allow developers to apply different levels of security to different parts of a XML document. However because of its point-to-point structure, SSL/TLS doesn't support chained services or workflow applications where user credentials can be passed through each stop in a transaction chain. (SSL/TLS also doesn't support the concept of an audit trail, in which it's possible to log who initiates the transaction at each step in the process.)

TLS runs on layers beneath application protocols such as HTTP, FTP, SMTP and NNTP, and above the TCP or UDP transport protocol, which form part of the TCP/IP protocol suite. While it can add security to any protocol that uses reliable connections (such as TCP), it is most commonly used with HTTP to form HTTPS. HTTPS is used to secure World Wide Web pages for applications such as electronic commerce. SMTP is also an area in which TLS has been growing and is specified in RFC 3207. These applications use public key certificates to verify the identity of endpoints.

HTTP is inherently insecure, all data is transmitted in plain text

- HTTPS is HTTP plus certificates
- HTTPS encrypts the entire message
- For web services bound to HTTP protocol, HTTPS in combination with WS-Security can be applied.

The Secure Socket Layer (SSL) is the most widely used protocol on the Internet. With SSL, two parties share a symmetric key and authentication is performed.

SSL creates a secure connection between a client and a server, over which any amount of data can be sent securely.

The protocol works as follows:

1. The client accesses a server
2. The server returns its certificate
3. The client prepares a random number that is a seed for generating a symmetric key
4. The client encrypts the seed number with a public key contained in the server certificate
5. The client sends the encrypted data to the server
6. The server decrypts the received data to extract the seed number



7. Both the client and server have the same seed number so they can generate a symmetric key from it

SSL has been approved by the Internet Engineering Task Force (IETF) as a standard.

3.2 SEAMLESS security platform functional approach

The current design of the security solution to the SEAMLESS project was made taking into account the use of the SEEMSEED project results and is depicted in the schematic in the next page.

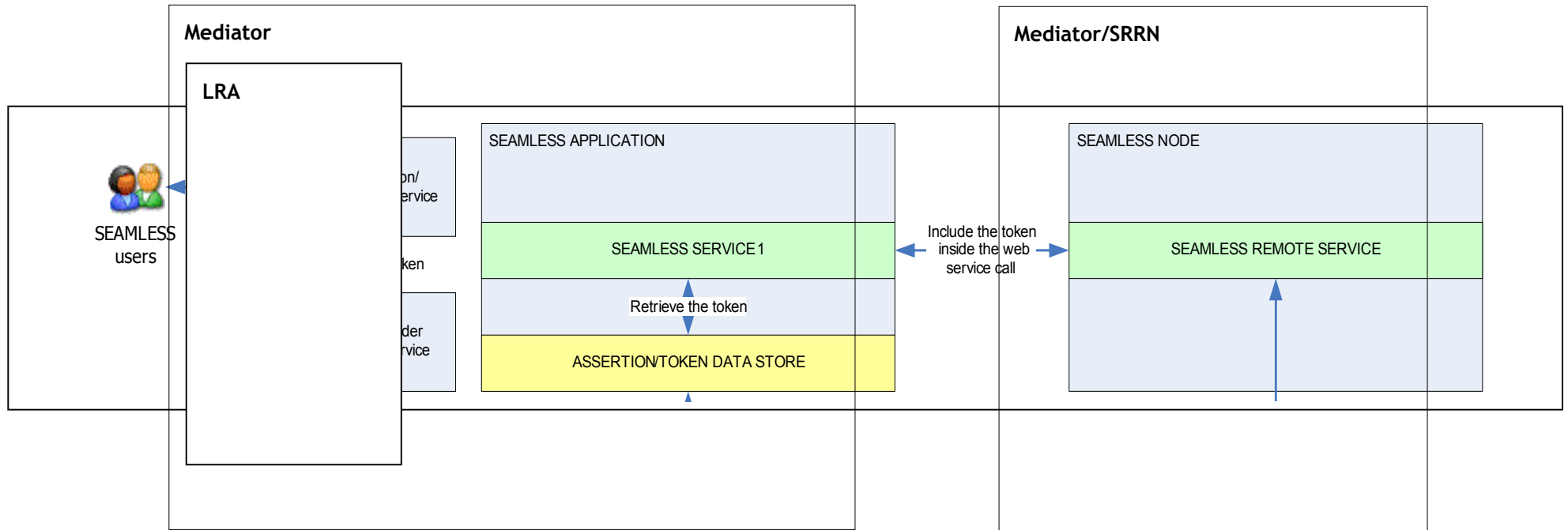
Basically during a session with the system, the user will authenticate to his home node (where the initial registration has taken place).

Following successful username and password matching or certificate authentication, the local identity provider (LRA) will generate a token/an assertion using the SAML protocol, with all information that has been identified as necessary to identify the user, including all credentials necessary for the services that are available and can be accessed. This token is thereafter uniquely identifying the user and needs to be sent to support security operations along with **every** request over the distributed SEAMLESS network.

Therefore it is required that the identity token is embedded in the implementation of all SEAMLESS services (as respective web service calls) according to the API defined below.

The universal endorsement of this approach both from service creators and service consumers allows for flexible and in fact distributed management of trust within SEAMLESS.





Security in the delivery of the platform services will be provided conceptually at two levels:

3.2.1 Transport level

Transport level defines the paths that are established for the communication between the nodes of the SEAMLESS network. Communication between the various nodes of the network should take place only using SSL and therefore using the https protocol.

The platform security framework takes care to ensure the respective implementation of node communication respects that all web service exchanges should be performed using the https protocol instead of http (non-secure – cf SSL/TLS overview).

3.2.2 Application level: authentication

The first step when establishing any connection between two entities is to authenticate themselves to each other, allowing them to demonstrate their identity. Alternative ways to establish one's identity to a service are:

- by user/password combination over SOAP can be used to authenticate, however, this is done in plain text (therefore, SSL / HTTPS need to be utilized to protect the credentials)
- Kerberos authentication service - Kerberos was created by MIT as a **solution to network security problems**. The Kerberos protocol uses **strong cryptography** so that a client can prove its identity to a server (and vice versa) across an insecure network connection. After a client and server have used Kerberos to prove their identity, they can also encrypt all of their communications to assure privacy and data integrity as they go about their business however
- SSL authentication, by using user digital certificates.

From the moment a user is authenticated however due to the distributed nature of the system and the request of SSO, we need the introduction of an **identity provider** as part of our determination to use also the SAML standard. And further we have to select also what topology to use for the authentication process. There are two main types of topologies (see figure below):

- Multilateral Model (centralised). The service providers with identity registries interact with a single Identity Provider.
- Bilateral Model (distributed). The service providers with identity registries, interoperating directly without a hub.

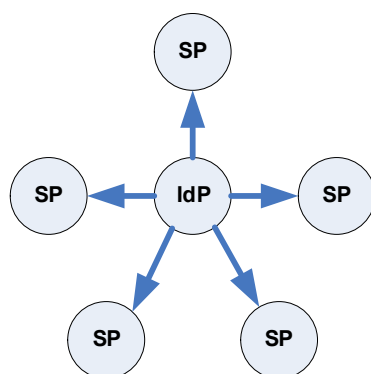


Figure 6 Single identity

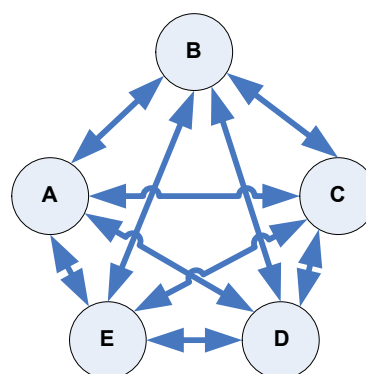


Figure 7 Multiple identities

Since besides the trust and security objective, the system should **not** be dependent on a single node, the identity provider (in order to fulfill also R2) should be used inside SEAMLESS. The idea is to have at each mediator site an identity provider server (IdP), as part of the Local Registration Authority identified before (Figure 5).

Following initial user authentication we will use SAML as described above for exchanging authentication and authorization data between an identity provider and a service provider.

Note, that in the current implementation of the project prototype it is however foreseen that the method of authentication can be changed during the project life cycle, in order to align the authentication technologies used along the infrastructure.

3.2.3 Application level: authorisation

Authorization has to do with the permissions that a certain user has on the SEAMLESS resources. The authorization will be handled in conjunction with the authentication functionalities described above. Due to the fact that SAML specifications allows to include inside the assertion selected attributes related to the current user, the roles of the user will be included in these assertions as part of the implementation of the SEAMLESS identity provider. The following extract illustrates how the 'role' attribute can be sent within the current assertion:

```
<saml:Attribute AttributeName="UserRole"
AttributeNameNamespace="urn:oasis:name:tc:SAML:1.0:assertion">
<saml:AttributeValue>superuser</saml:AttributeValue>
</saml:Attribute>
```

SAML assertions are transferred from the identity providers to service providers. Assertions contain statements that service providers use to make access control decisions. These are the three types of statements provided by SAML:

Authentication statements

Authentication statements assert to the service provider that the principal did indeed authenticate with the identity provider at a particular time using a particular method of authentication. Other information about the principal may be disclosed in an authentication statement. An example of such of statement is provided below:

```
<saml:Assertion
xmlns:saml="urn:oasis:names:tc:SAML:1.0:assertion"
MajorVersion="1" MinorVersion="1"
AssertionID="..."
Issuer="https://idp.org/saml/"
IssueInstant="2005-03-21T19:56:10.250Z">
<saml:Conditions
NotBefore="2005-03-21T19:51:10.250Z"
NotOnOrAfter="2005-03-21T20:01:10.250Z"/>
<saml:AuthenticationStatement
AuthenticationMethod="urn:oasis:names:tc:SAML:1.0:am:password"
AuthenticationInstant="2005-03-19T19:56:10.102Z">
<saml:Subject>
<saml:NameIdentifier
Format="urn:oasis:names:tc:SAML:1.1:nameid-format:emailAddress">
user@seamless-eu.org
</saml:NameIdentifier>
<saml:SubjectConfirmation>
```

```
<saml:ConfirmationMethod>
  urn:oasis:names:tc:SAML:1.0:cm:artifact
</saml:ConfirmationMethod>
</saml:SubjectConfirmation>
</saml:Subject>
</saml:AuthenticationStatement>
</saml:Assertion>
```

Attribute statements

An attribute statement can indicate whether or not the principal has an affiliation of "superuser", which the service provider then uses to allow or deny access to a certain SEAMLESS application service:

```
<saml:Assertion
xmlns:saml="urn:oasis:names:tc:SAML:1.0:assertion"
MajorVersion="1" MinorVersion="1"
Issuer="https://www.seamless-eu.org/saml/" ...>
<saml:Conditions NotBefore="..." NotAfter="..." />
<saml:AuthenticationStatement
  AuthenticationMethod="urn:oasis:names:tc:SAML:1.0:am:X509-PKI"
  AuthenticationInstant="...">
  <saml:Subject>...</saml:Subject>
</saml:AuthenticationStatement>
<saml:AttributeStatement>
  <saml:Subject>...</saml:Subject>
  <saml:Attribute
    AttributeName="urn:mace:dir:attribute-def:eduPersonScopedAffiliation"
    AttributeNamespace="urn:mace:shibboleth:1.0:attributeNamespace:uri">
    <saml:AttributeValue Scope="seamless-eu.org">
      user
    </saml:AttributeValue>
    <saml:AttributeValue Scope="seamless-eu.org">
      superuser
    </saml:AttributeValue>
  </saml:Attribute>
</saml:AttributeStatement>
</saml:Assertion>
```

Authorization decision statements

In the above sample showing how a 'superuser' might obtain access to a SEAMLESS application, the security service provider is functioning as both a policy enforcement point and a policy decision point. In some situations, it may be preferable to associate the policy decision point with the identity provider. In this case, the service provider passes a URI to the identity provider who asserts an authorization decision statement that dictates whether or not the principal should be allowed access to the secured resource at the given URI. The following represents an authorization decision sample.

```
<saml:Assertion
  xmlns:saml="urn:oasis:names:tc:SAML:1.0:assertion"
  MajorVersion="1" MinorVersion="1"
  Issuer="https://www.seamless-eu.org/saml/" ...>
  <saml:Conditions .../>
  <saml:AuthorizationDecisionStatement
    Decision="Permit"
    Resource="https://www.seamless-eu.org/*****.html">
    <saml:Action>read</saml:Action>
    <saml:Subject>...</saml:Subject>
  </saml:AuthorizationDecisionStatement>
</saml:Assertion>
```

User login is supported through the Authentication/Authorization service, which (in case of success) will call the Identity Provider Service. This service will allow the user/application obtain an assertion which will then be stored in the SEAMLESS application inside a security data holder. Following that, whenever a call is made to the SEAMLESS Services which needs to make additional calls, this assertion will be attached (as discussed above) in the next sections) to the respective call and the additional information (string or documents) can be encrypted and digitally signed using the Encryption service.

3.2.4 Application level: data integrity and non-repudiation

In order to make sure that an item (query, document, etc.) belongs to the sender/author and the content has not changed, we will use digital signatures.

The technology which will be used is XML Signature, following the W3C standard specification. Because integrity of information is closely correlated to data confidentiality, a detailed description of the respective approach is provided in the next part. The operations which will be made available will allow to sign a document or a string and to check a signature.

3.2.5 Application level: data confidentiality

To support data confidentiality (and data integrity) the application will make use of encryption.

In the context of the SEAMLESS project in order to use symmetric encryption we need to have a common secret shared by the two parties who are communicating.

Because the process of sending information is not always direct (meaning that a user could store a document inside a SEAMLESS repository, and the recipient is allowed to retrieve it at any point in time later on), the preferred solution is to use asymmetric encryption (using private and public keys).



Following that reasoning, in designing the security components we started from one simple assumption: *when a command/query is issued it always has a sender*. The recipient is either intended to be a particular entity or a group of entities.

Due to the fact that encrypting and decrypting files is more time consuming than the symmetric algorithms, we have decided to use a combined approach as follows:

1. A key is generated using a symmetric algorithm. The data to be encrypted is encoded using this key.
2. The above key is encrypted using the asymmetric algorithm.
3. The encrypted file is merged with the key encrypted together with the length of the key
4. This result of the merger is sent over the network

We present below the flow of control in the various cases that can appear during use of the platform:

Case 1. One specific recipient

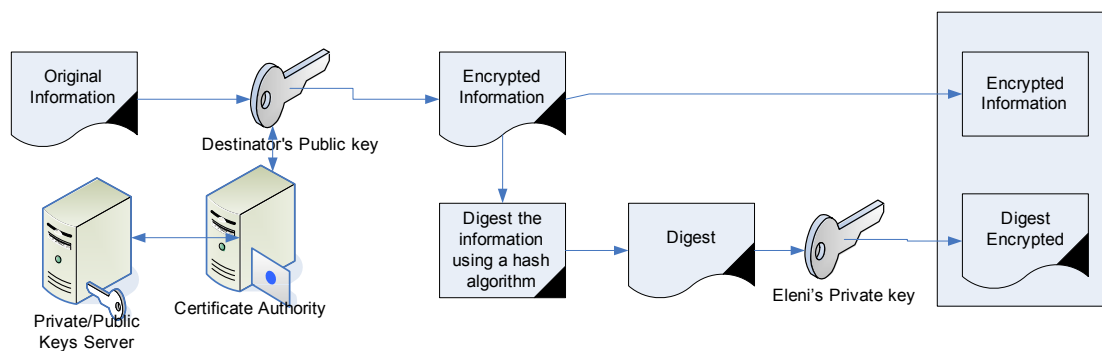


Figure 8. Security encapsulation in Send Data Operation- One sender –to- One receiver

In the above case only the recipient is able to read the respective information, being certain of the identity of the sender too. On the receiver end the reverse process is taking place.

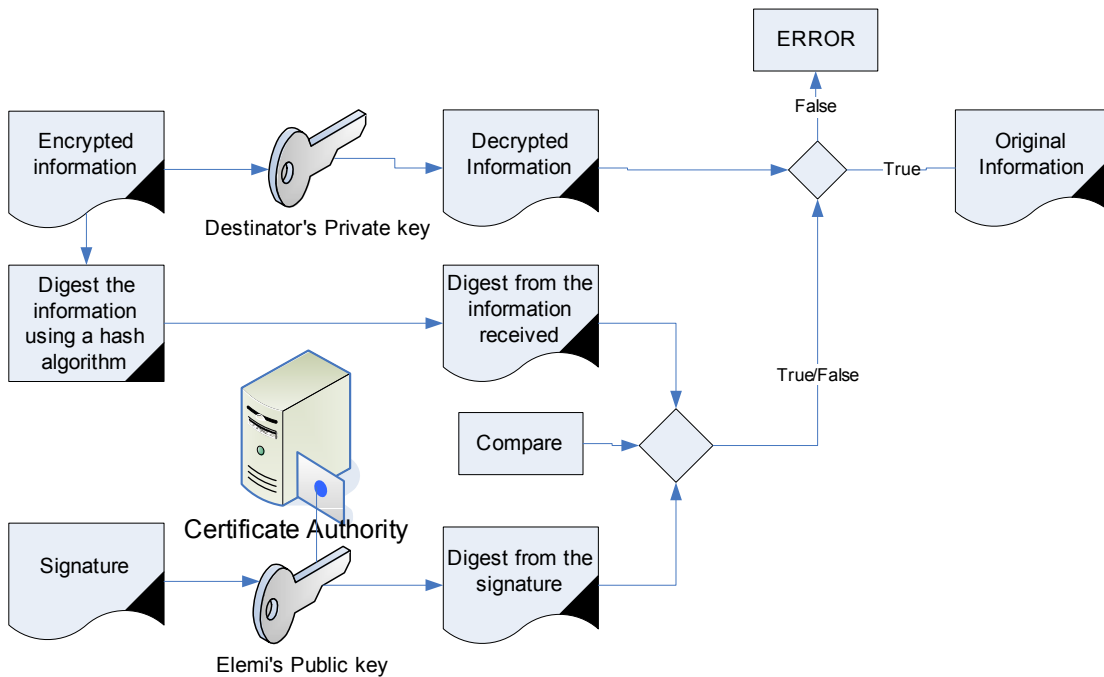


Figure 9. Security de-encapsulation in Receive Data Operation- One sender –to- One receiver

Case 2. Group of recipients/all authenticated

In this case the information will be encrypted with the sender private key in order to be able to be decrypted by any entity having the sender public key.

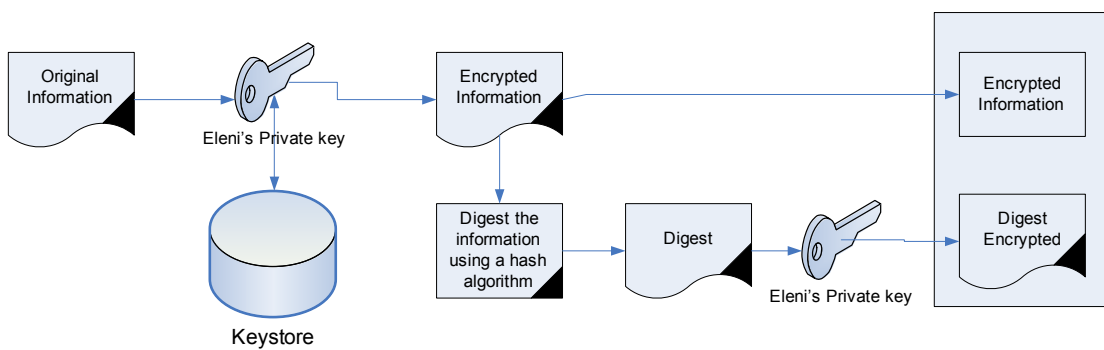


Figure 10. Send Data Operation- One Sender – Multiple Receivers

The figure bellow presents the process which takes place when the information is received.

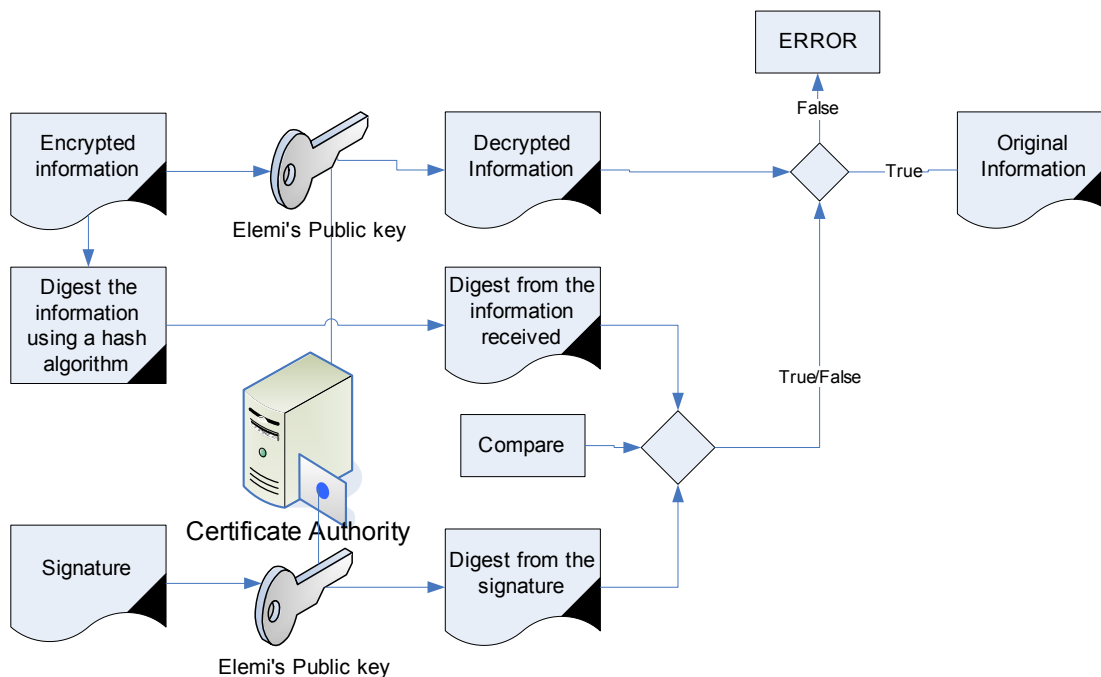


Figure 11. Receive Data – One sender – Multiple Receivers

In order to distinguish between the messages which are sent to all recipients, which means public, and the ones which are sent to a specific number of recipients we need to be able to distinguish the messages into two categories: private messages and public messages.

The actual implementation makes this distinction based on the assumption that public documents need not be encrypted.

In order to allow the documents to be also encrypted and signed, the respective implementation will work on base64 based strings. In the base64 case, once the document will be available at the SEAMLESS application layer, it will be encrypted using an asymmetric algorithm and then signed appropriately.

In conclusion the above presented method when adopted within the framework of SEAMLESS will ensure:

- data integrity and the sender is the one who claims because only him could poses the private key and digest information is the same
- Confidentiality is ensured using this method because only the recipient could decrypt the encrypted information using his private key.

3.3 Pilot implementation approach

The following sections will present the infrastructure needed for the security module to work properly and the software components in terms of description and APIs.

It is clear from the above flows that we need an internal mechanism for managing the keys and certificates. Because the certificates need to be trusted only by the users within the SEAMLESS client community, it is within the mediator that is proposed to maintain the respective infrastructure that will have this responsibility. At each mediator there is a set of components that have to be made available in order to provide support for the implementation of platform security. These components have been identified tested in practice and adjusted to provide stable operation and are presented in the next paragraph.

3.3.1 Infrastructure

The detailed implementation of the security functionality has had as broad guidelines the following two requirements

- - be as much as it is possible independent from the rest of the system
- - use open source software and components available for the prototype.

In taking the decisions how to implement the functionalities mentioned above we also tried purposely to avoid introducing significant performance overheads to the system operations.

In this aspect the idea to implement the encryption operation as a web service was dropped, since if it were to be implemented as a web service it will increase the time necessary for a simple operation (and eventually raise other security issues).

Thus, the current implementation of the security mechanisms within the SEAMLESS framework is composed by

- An authentication & authorization web service (implemented using Axis on top of a Tomcat 5.5 server) and
- Java components which will ensure the data integrity and confidentiality and are available for the applications that will run in the mediator.

The following software applications, components and hardware elements are also necessary to be installed at the respective nodes in order to support the implementation of the platform security:

- **Database/User repository** This database will maintain the existing users and their associated roles in the application and should be the same as the database used by the SEAMLESS application. The choice will be mainly determined by the open source nature of the products. (possible solutions that have been evaluated and can be supported being MySQL or PostgreSQL databases).
- **Private/Public key server.** This server will manage users' keys. In the general case such a PKI server is associated with a trusted independent certificate authority (CA). Since in our case key functionality is to be internally supported at the mediators, we need to have the respective server software at each site. Following an internal evaluation of alternative implementations we eventually select to use an already existing solution called EJBCA, available as open source software under the GPL license. This solution has been tested and adapted to provide a stable infrastructure element of the SEAMLESS prototype security functionality.
- **XKMS server.** Due to the fact that most open source implementations for XKMS server available as open source when thoroughly tested have proven to be difficult to support and some time unstable, different alternatives are being studied to make available certificate management functionality.
- **Identity server provider** (SAML compliant). This server is needed to implement the Authentication & Authorization functionalities.
- **Servlet container** The server will host the web services which will be developed (with the most probable choice being **Apache Tomcat Server** already used in the prototype).
- **Web Service provider.** An implementation of web services, which will be used in conjunction with the servlet container. Research was carried out using the Axis implementation for the web service provider. Web services will serve mainly for the initial authentication since then the mediator infrastructure can maintain a local keystore.

The following figure shows the overall architecture extended to include the components pertinent to the security functionality implementation:

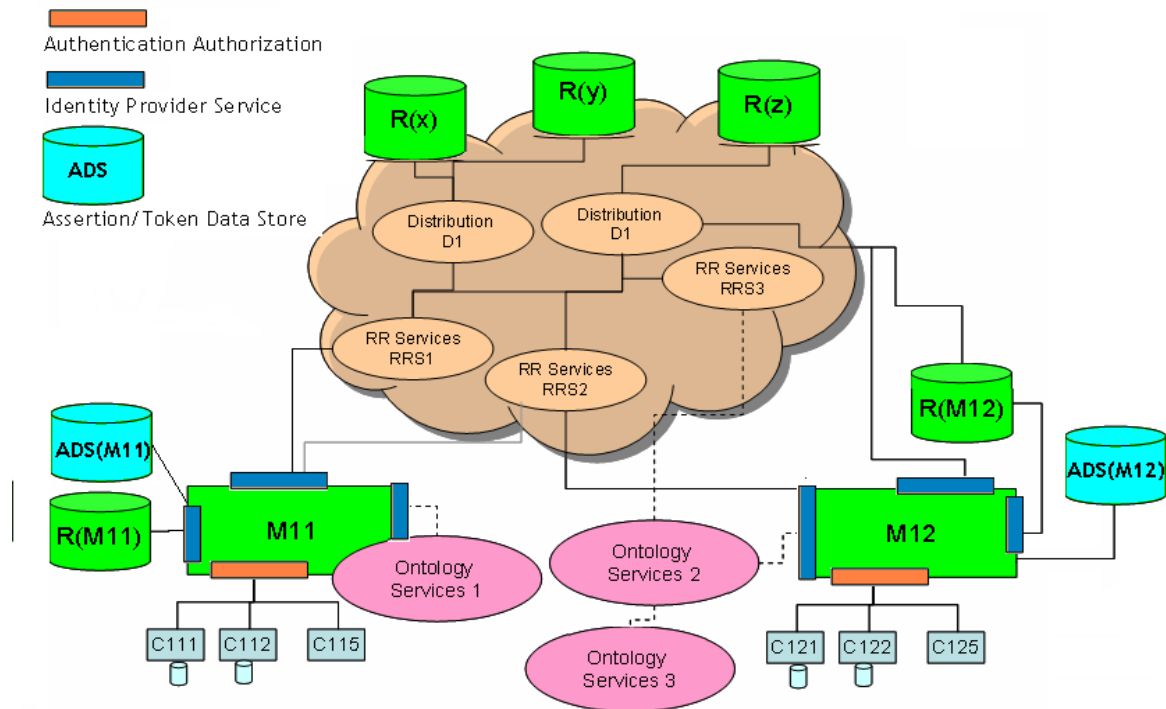


Figure 12 SEAMLESS Architecture with Security Components

3.3.2 Software modules: AAInterface

This and the following sections will describe the classes and methods available in different components and in the respective prototype in D3.2.1.

The authentication & authorization web service is called AAInterface.

In the following paragraphs we describe shortly the operations to be implemented and a complete WSDL file is provided along with this document.

We have chosen for the current implementation the Single Sign On principle based on SAML assertion. The initial authentication will be made using a user and a password. Following that, the identity server provider will issue an assertion. This assertion will be passed to all the calls of the methods. The way it will be integrated is using an extra component available at the web service client and server.

This module provides the authentication and authorization functionalities. The way it is accessed by other applications is by web services. The methods are presented in the following paragraph and made also available as wsdl files. The following methods can be changed or updated with new ones as we progress with the development.

The authorization functionalities are to be implemented also in this component and the actual user attributes will be stored inside the user token (SAML assertion).

Components

Class name	AAInterface
Methods	Description
Token AAQuery(string username, string password)	This method will allow the user to authenticate based on a user and a password and it will return a token (SAML

	assertion).
Boolean validateToken(token userToken)	This method will receive as input a user token and it will return true or false based on the fact the token is good or not.
Boolean isAllowed(token userToken, string resource)	This method will receive as input a user token and a resource identified by <i>resource</i> and will return true or false based on the fact that the user is allowed to access that resource.

Optionally, in order to manipulate credentials the following design pattern can be used

Assertion Builder

It creates SAML assertion statements.

Description of Assertion Builder

It encapsulates the processing control logic in order to create SAML

- authentication statements
- authorization decision statements
- attribute statements.

Under the proposed SSO environment the client authenticates to the source site and may request access to a resource form destination site. After successful authentication the source site will redirect the client request to the destination site assuming that the source site identified that the client is allowed to access the destination site. After that, the destination site will issue a SAML request to ask for authentication assertion from the source site. Assertion Builder will be used to encapsulate sign on information and user credentials to generate SAML assertion statements. The destination site will respond to the client for resource access. It will also handle authorization decisions and attribute statements to determine what access level is allowed for client request.

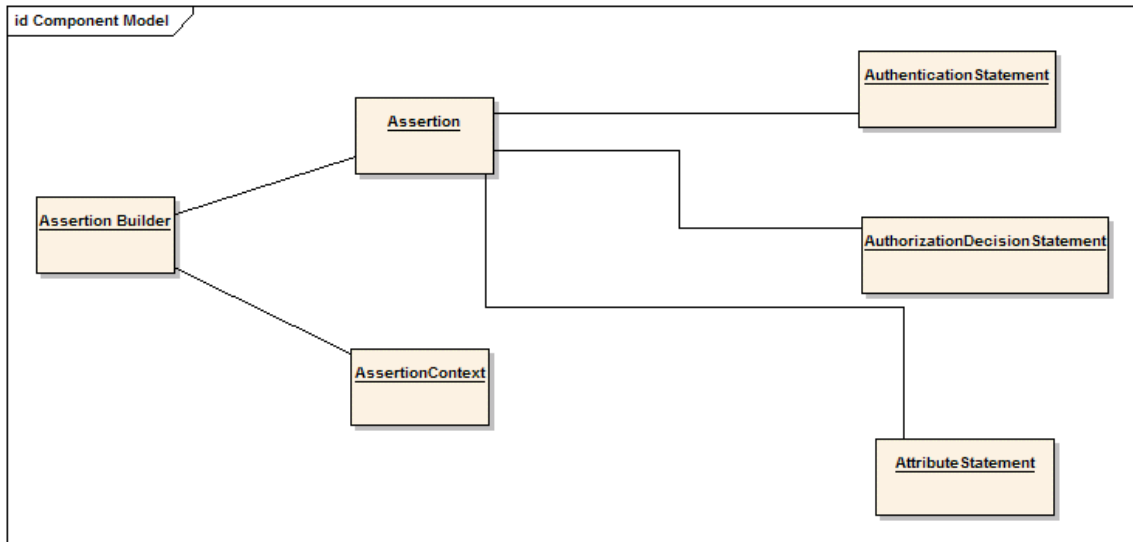


Figure 13 Assertion Builder components

In this scenario the Assertion Builder was designed to provide the service of generating authentication, attribute and authorization decision statements. It creates an Assertion Context and makes an Assertion which can be of one of these 3 types.

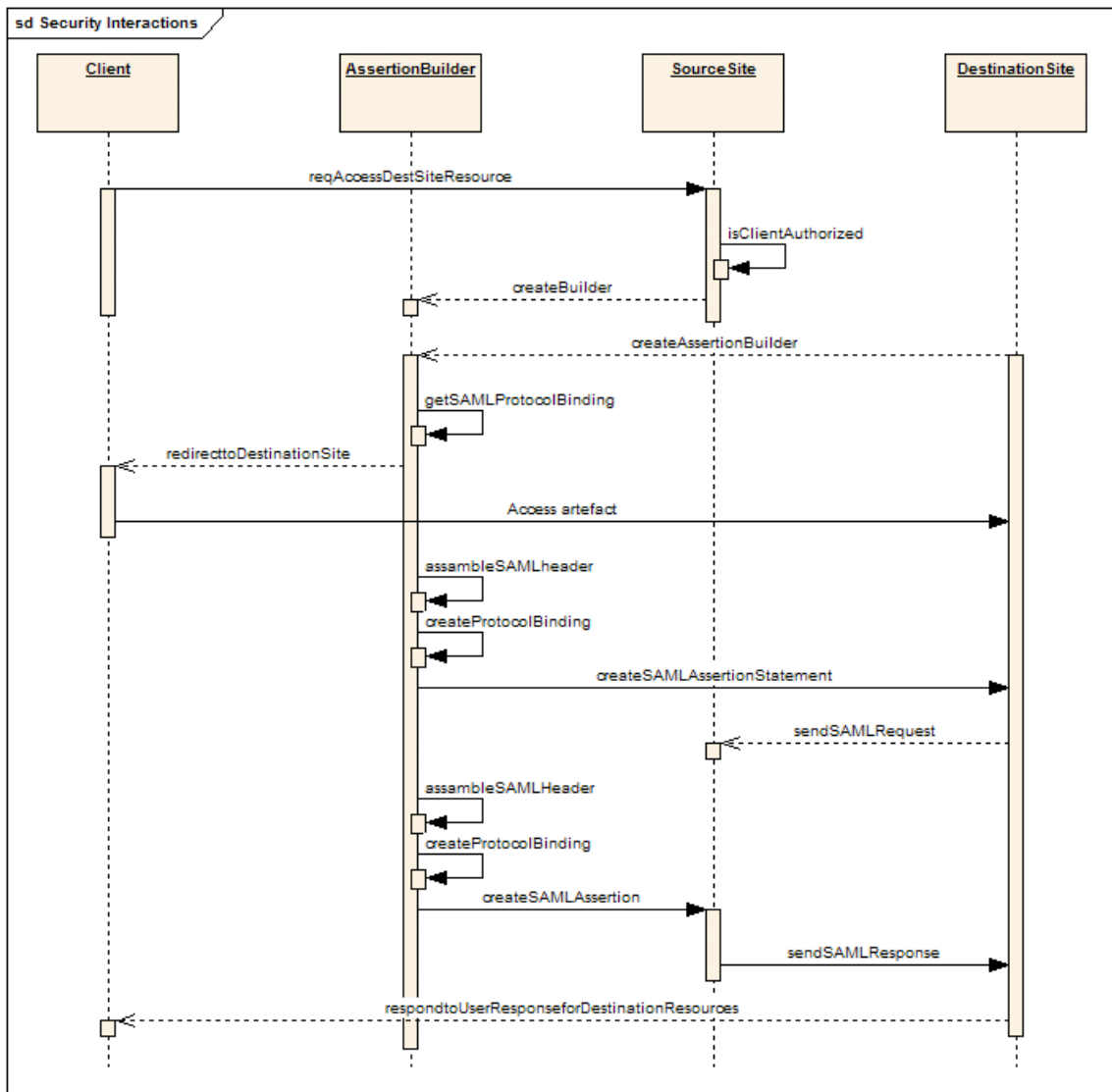


Figure 14 Assertion Builder sequence diagram

In this scenario a client is requesting a resource from destination site via source site which is seen as a single sign on service provider

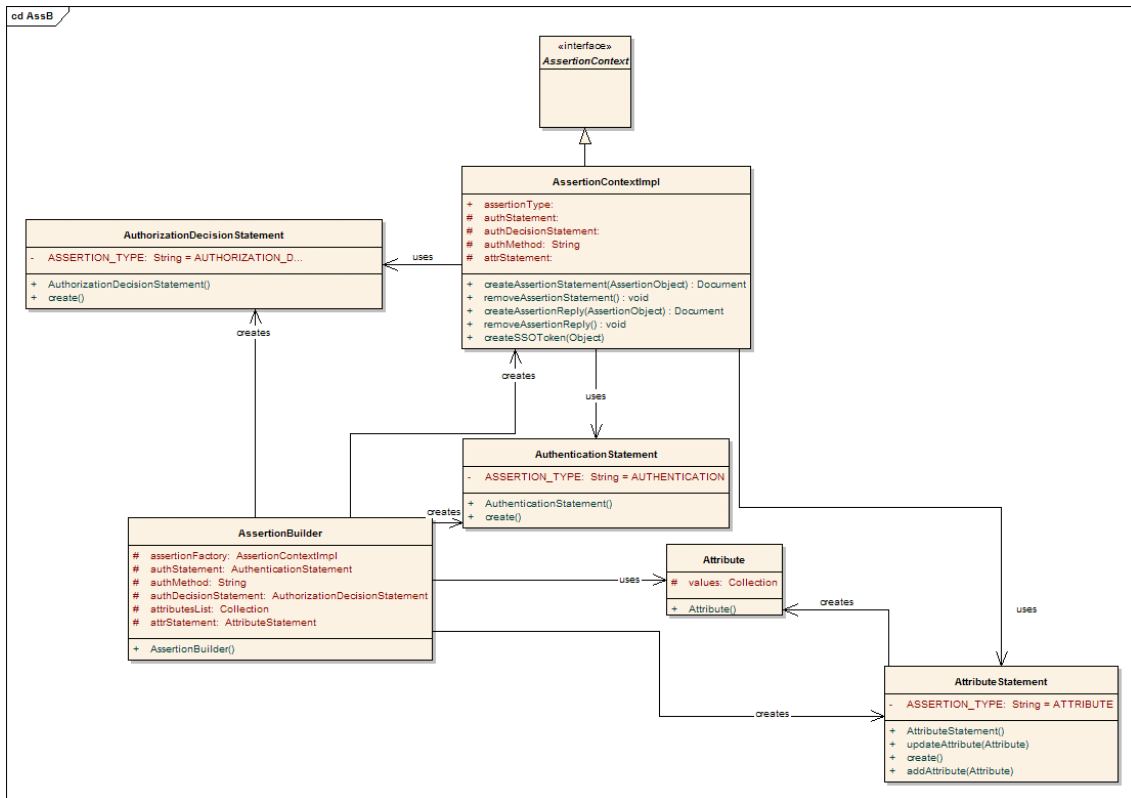


Figure 15 Assertion Builder Class Diagram

Description

AssertionContext interface for managing the system context when creating SAML assertion statements, SAML assertion types

AssertionContextImpl implementation of *AssertionContext*

Assertion

AuthenticationStatement extension of *Assertion*

AuthorizationDecisionStatement extension of *Assertion*

AttributeStatement extension of *Assertion*

Attribute – encapsulates characteristics of subject and denotes the attributes in *AttributeStatement*

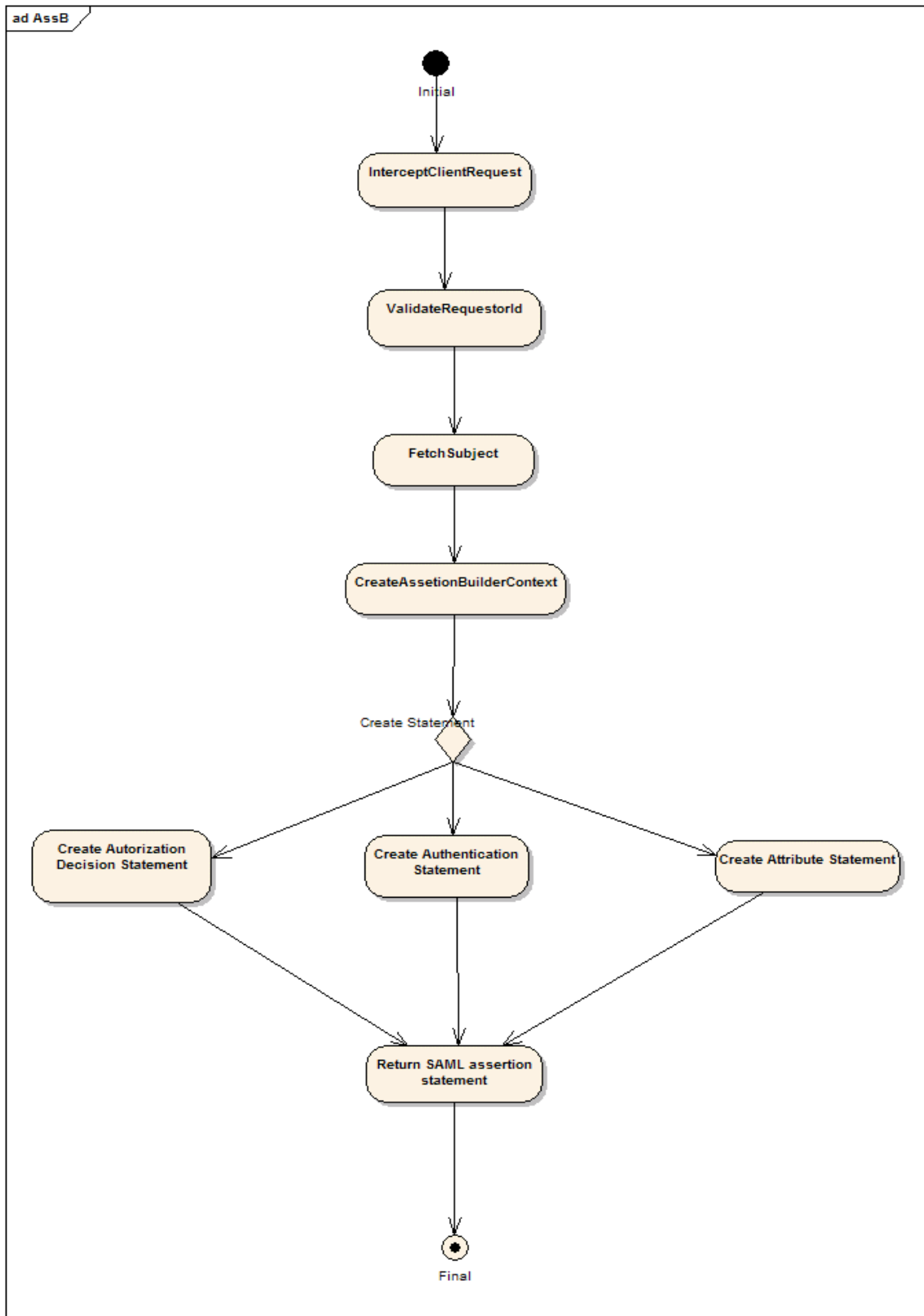


Figure 16 Assertion Builder Activity Flow Diagram

3.3.3 Software modules: SecureIT

This module will be in charge of the signature and encryption operations. This module will make the necessary connection to retrieve various users' public keys.

The SecureIT operations will be applied also on binary data. That is why we chose to make the signature available separately from the document when it was also a possibility to include the binary data encoded as base64 in a XML file together with the signature and encode it again to obtain a single base64 variable. However, in order to avoid overheads we make the decision shown above.

This component will deal with signing and encrypting functionalities, but also with key management. The exposed methods related to key manipulation hide the complexity of this task.

Components

Class name	Signature
Methods	Description
byte[] sign(string EncryptionMethod, string Key, byte[] document)	The method will provide as output the signature of the document <i>document</i> using the key <i>Key</i> and method <i>EncryptionMethod</i> .
boolean verify(byte[] signature ,byte[] document)	The method will provide as output a boolean result of the signature verification against the document <i>document</i> . True if everything is ok, false otherwise.

Class name	Encryption
Methods	Description
Byte[] encrypt(string EncryptionMethod, string Key, byte[] document)	The method will provide as output the encrypted version of the document using the method <i>EncryptionMethod</i> and the key <i>Key</i> .
Byte[] decrypt(string EncryptionMethod, string Key, byte[] document)	The method will provide as output the decrypted version of the encrypted document using the method <i>EncryptionMethod</i> and the key <i>Key</i> .

Class name	FileUtils
Methods	Description
Byte[] loadFile(string path)	The method will load the document into the memory from the path <i>path</i> .
saveFile(string path, byte[] Document)	The method will save the document from the memory on the file system using the path <i>path</i> .

Class name	KeyUtils
Methods	Description
Key loadKey(string userid)	The method will locate and load the key of the user identified by <i>userid</i> .
Certificate loadCertificate(string userid)	The method will locate and load the certificate of the user identified by <i>userid</i> .

For encryption in the prototype we considered the following algorithms



- For asymmetric encryption we considered *RSA*
- For symmetric encryption we considered *DES* or *RC4*.

For signing we had in mind the following algorithms: *SHA1withDSA*.

For hashing we had in mind the following algorithms *MD5*, *SHA1*, *RIPE_MD*.

3.3.4 Software modules: Logger

All trustworthy applications require a secure and reliable logging capability. This logging capability may be needed for forensic purposes and must be secured against stealing or manipulation by an attacker. Logging must be centralized on the Mediator to avoid redundant code throughout the code base. Integrity and confidentiality of captured data may be protected by applying encryption algorithms.

The Secure Logger design pattern is the best match in this case. It provides centralized control of logging functionality that can be used in various places throughout the application request and response. It takes care of how events are logged in a reliable and secure manner

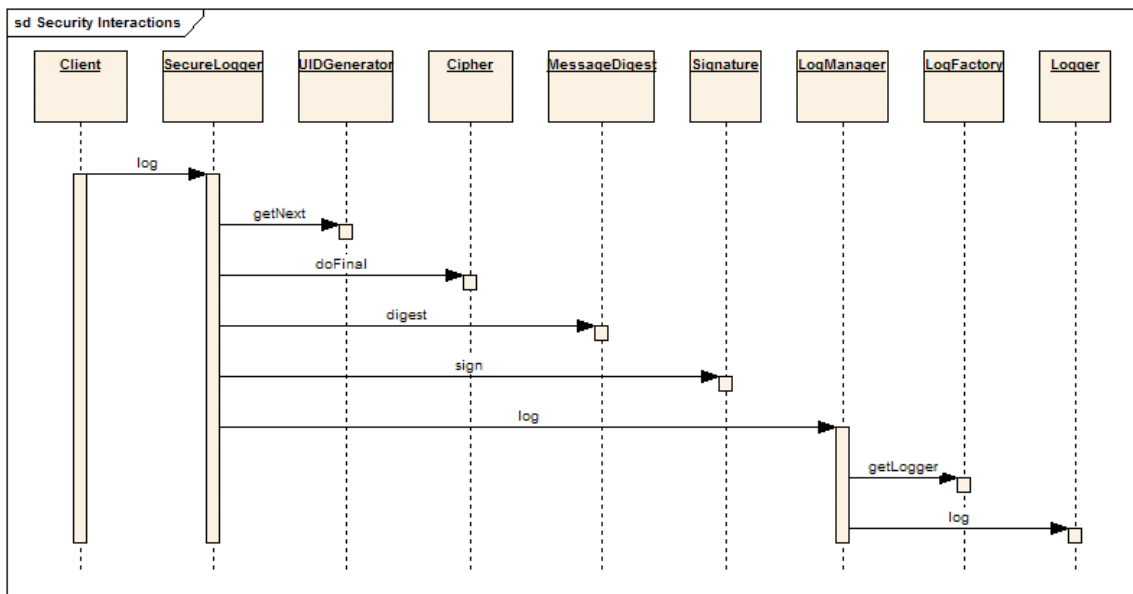


Figure 17 Secure Logger Sequence Diagram

There are two parts to this logging process. The first part involves securing the data to be logged (using **SecureIT** API) and the second part involves logging the secured data. The SecureLogger class takes care of securing the data and the LogManager class takes care of logging it.

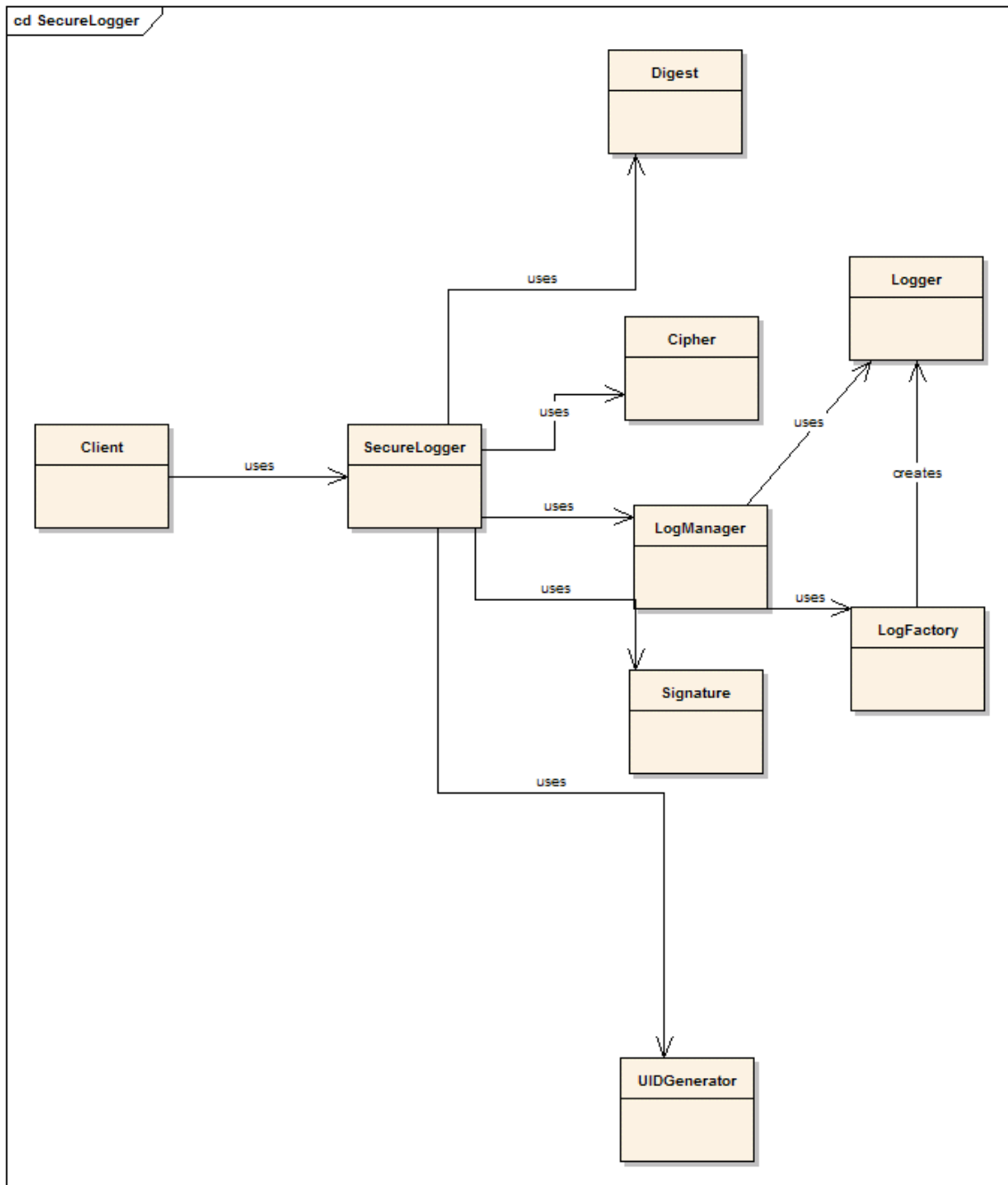


Figure 18 Secure Logger Class Diagram

Description:

Secure Data Logger forces preprocessing of the data prior to logging it. After the data is secured in the preprocessing, it is sent to the logger in the usual manner.

We use the **MessageDigest**, **Cipher**, **Signature**, and **UIDGenerator** classes for applying cryptographic mechanisms and performing various functions necessary to guarantee the data logged is confidential and tamperproof.

The sensitive data are encrypted by using symmetric key algorithm.

Data alteration can be prevented by using digitally signed message digests. A message digest is generated for each message in the log file and then signed. This way we prevent modification of message and creating another message digest for modified data.

We use sequence numbers to detect deletion of data. Each entry in the log must have a sequence number that will be part of the data that gets signed. This way gaps between messages will easily be discovered.



4 Interaction example with the SRRN

This chapter will try to give an example how the interface with the existing systems can be performed based on the technical approach that has been outlined before.

We will take as example the web service method *storeRawDocument*.

4.1 From signature & encryption point of view

The current implementation shows the following description:

Parameter Name	Parameter Type
Document	Base64Binary
Version	String
transactionID	String

From the client application which will call the above method, the Document variable will contain a file represented as base64. In order to update the implementation in order to include the security module calls the client should use the following code segment:

```
byte[] docData = loadFile(. . .);
// EncMeth=encryption method, Key=user's key
byte[] docEncrypted = encrypt(EncMeth, Key, docData);
byte[] docEncryptedSignature = sign(docEncrypted);

//call storeRawDocument method
storeRawDocument(docEncrypted,version,transactionID,docEncryptedSignature)
```

From the above example it is clear that in order to comply with this call the server side implementation has to provide an overloaded method *storeRawDocument* which will have as parameters the following:

Parameter Name	Parameter Type
Document	Base64Binary
Version	String
transactionID	String
Signature (New)	Base64Binary

Inside this new method the following operations should take place:

```
if (verify(Signature,Document)) {
    byte[] mydoc = decrypt(EncMeth, Key, Document);
```

```
} else {  
    //the document is corrupted/not safe  
}  
  
// call the initial method  
storeRawDocument(Document, Version, transactionID);
```

This way the effort adapting existing methods is minimized as much as possible.

4.2 From authentication & authorization point of view

Based on the work carried out during this period it proves that this is the most sensitive part of the integration. Research and final selection was made on the assumption that the communication between different nodes is done by web services.

Once the security assertions/tokens are generated they must be included in every web service call. To do that we need to use an implementation of the WS-Security specifications. WSS4J is such a library which was used during our tests with limited success and needed significant adjustments to become stable and usable.

In order to secure the existing or new web services the respective configuration file should be updated. Some interoperability issues between the .NET and Java implementation of the web services can appear because the format of this specifications is different.

There are two parts to process SAML assertions in a web service:

- Check the low level embedding of the SAML assertion inside the soap headers
- Check the authorization statements inside the assertion

However, WSS4J does not know how to natively process SAML assertions, so we have to break it into two:

- Signature verification
- Actual assertion processing

The signature verification checks to make sure the signature on the soap message was signed by a certificate issued by a trusted certificate authority (in this case is the SEAMLESS organization).

The SAML assertion processing is limited to packing the assertion into the message context and letting you retrieve it from your call.

The configuration to be made in the java web services area (server side) is the following:

```
<service name="Receiver" provider="java:RPC" attachments="DIME">  
    <requestFlow>  
        <handler type="java:org.seamless.wssec.CachedDoAllReceiver">  
            <parameter name="action" value="Timestamp Signature  
SAMLTokenUnsigned"/>  
        </handler>  
    </requestFlow>  
    <parameter name="className" value="org.seamless.receiving.Receiver"/>  
    <parameter name="allowedMethods" value="*" />  
    <namespace>urn:receiving.seamless.org</namespace>
```

```
</service>
```

The CachedDoAllReceiver is provided as a separate component and it needs to be included in the lib area of the respective web service.

In order to correctly provide the security information the client calls should include also the SAML info.

To do that in the context of the client the example tested is using the StockQuoteService from the Axis distribution. The following code is the normal code used to call a web service when the client components have been created using the wsdl2java utility.

```
1. StockQuoteServiceServiceLocator locator = new StockQuoteServiceServiceLocator();
2. StockQuoteService service = locator.getStockWss01();
3. float quote = service.getQuote(args[0]);
```

In order to add the security information inside the soap message the following code should be inserted after the first line:

```
EngineConfiguration clientConfig=createClientConfig();
locator.setEngineConfiguration(clientConfig);
locator.setEngine(new AxisClient(clientConfig));
```

“EngineConfiguration createClientConfig()” represents a method which is provided as a component to be included in the lib path of the client application.

5 Conclusions

The security approach for SEAMLESS has been presented, based on the proposed design and existing SRRN infrastructure. Additional research into combination of available technologies taking into consideration the distributed environment of SEAMLESS allowed several alternatives to be examined and tested. The final result is a set of components adapted from available open source solutions and an implementation of prototype Java components that address security requirements set within the overall design of the SEAMLESS platform and handle effectively security issues for the deployment of the pilot services.

Improvements of course can always be made and to this extend refinements and further development has been provisioned.



Annex I – SEEMseed (SRRN) Security Support

The current SRRN implementation provides both access control and other security features. All these mechanisms are based on Digital certificates, SSL and PKI technologies, This ensure:

- The identity of the user that is accessing any kind of resource at the servers,
- The server (node) identity itself.

The SEEMseed security infrastructure (Figure 23) is based on a Public Key Infrastructure to achieve secure communication between businesses. It is also used to provide authentication and authorization using digital certificates. The Server certificates are used for Secure Sockets Layer (SSL) protocol communication which also provides confidentiality. Digital Signatures, using the XML DSig technology, are used to provide integrity and non-repudiation for the objects management used in the SRRN, as well as user authentication.

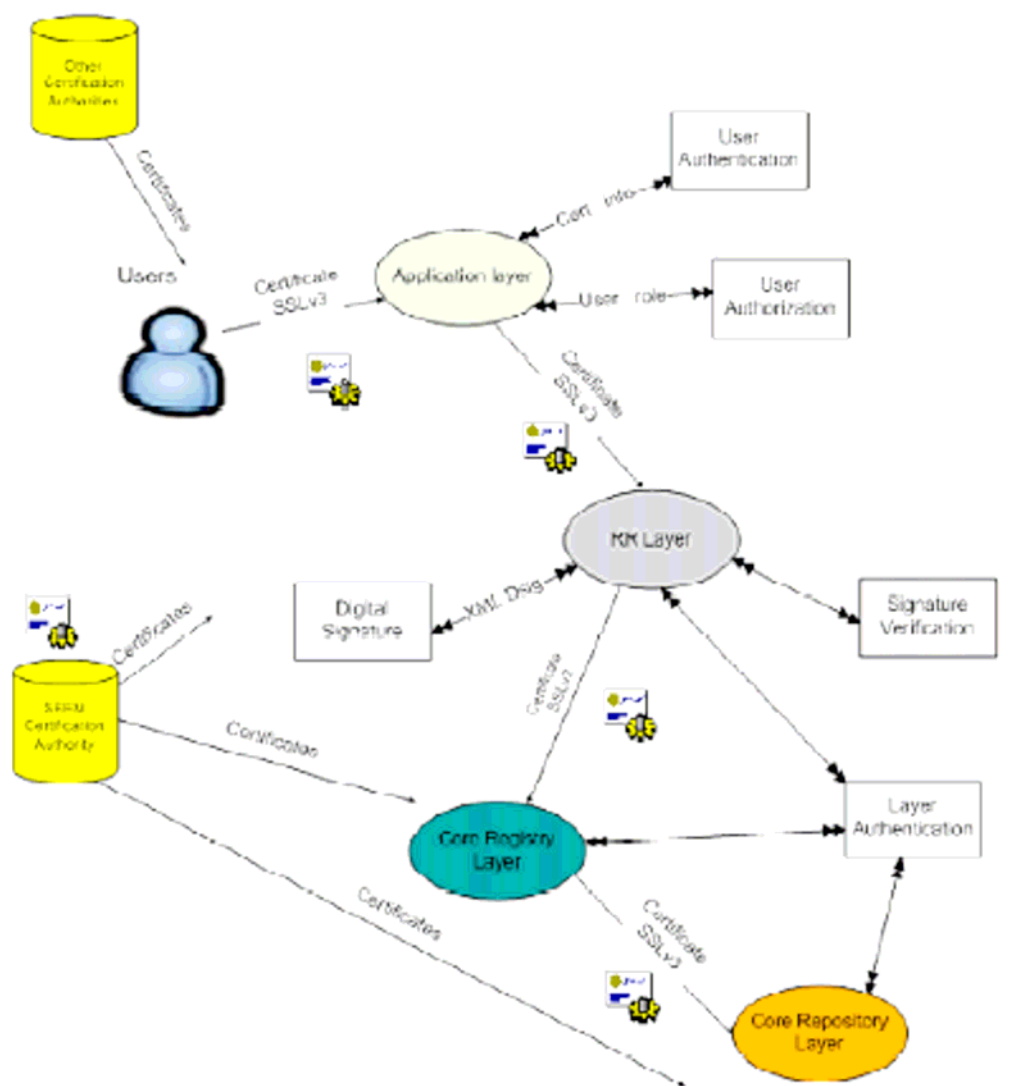


Figure 13: The SEEMseed security infrastructure

Access control is implemented at different SRRN levels, with a pre-authentication feature at the RRService Layer, and authentication plus access control at the Core Repository layer. The Distribution layer does not have a requirement to implement access control features.

An important aspect in an infrastructure that so distributed as the SRRN is, is the mutual confidence between all the nodes that composes the network of the registries and repositories. In the SRRN, all the nodes make use of the SSL (Socket Secure Layer) protocol for the encryption of the information transferred, and certificate authentication for mutual server recognition. So each registry and repository keeps a list for all the certificate authorities it recognizes. The SEEM certification authority is recommended as certificate registrar for all the nodes that composes the infrastructure (although it is not mandatory).

The authentication mechanism between servers is used along all the chain of queries that an operation (write or query) implies, and that authentication mechanism is embedded within the communications protocol itself. SSL natively allows only the usage of one certificate, but that mechanism (very valid for the nodes that compose the SRRN network) has also the constraint of being dependent on the confidence chain of certificate authorities. So a more flexible mechanism is needed (i.e. not currently within the SRRN) for user authentication. In SRRN, user authentication is based on the validity of the signature of the user (or the application in name of the user). So, a digital signature (XMLDSig) of the credentials of a user (including the certificate itself -public part-) is transferred on each call, so that the repositories could check the validity, and if there is any kind of access right for this certificate in the specific in the data stored on it

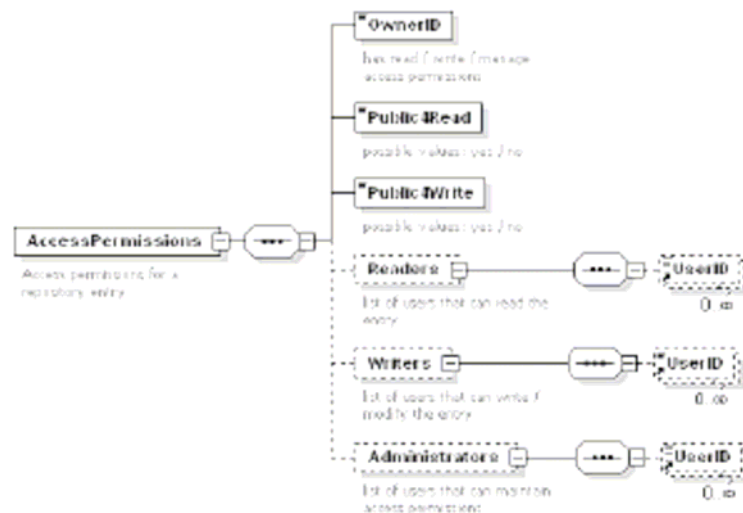
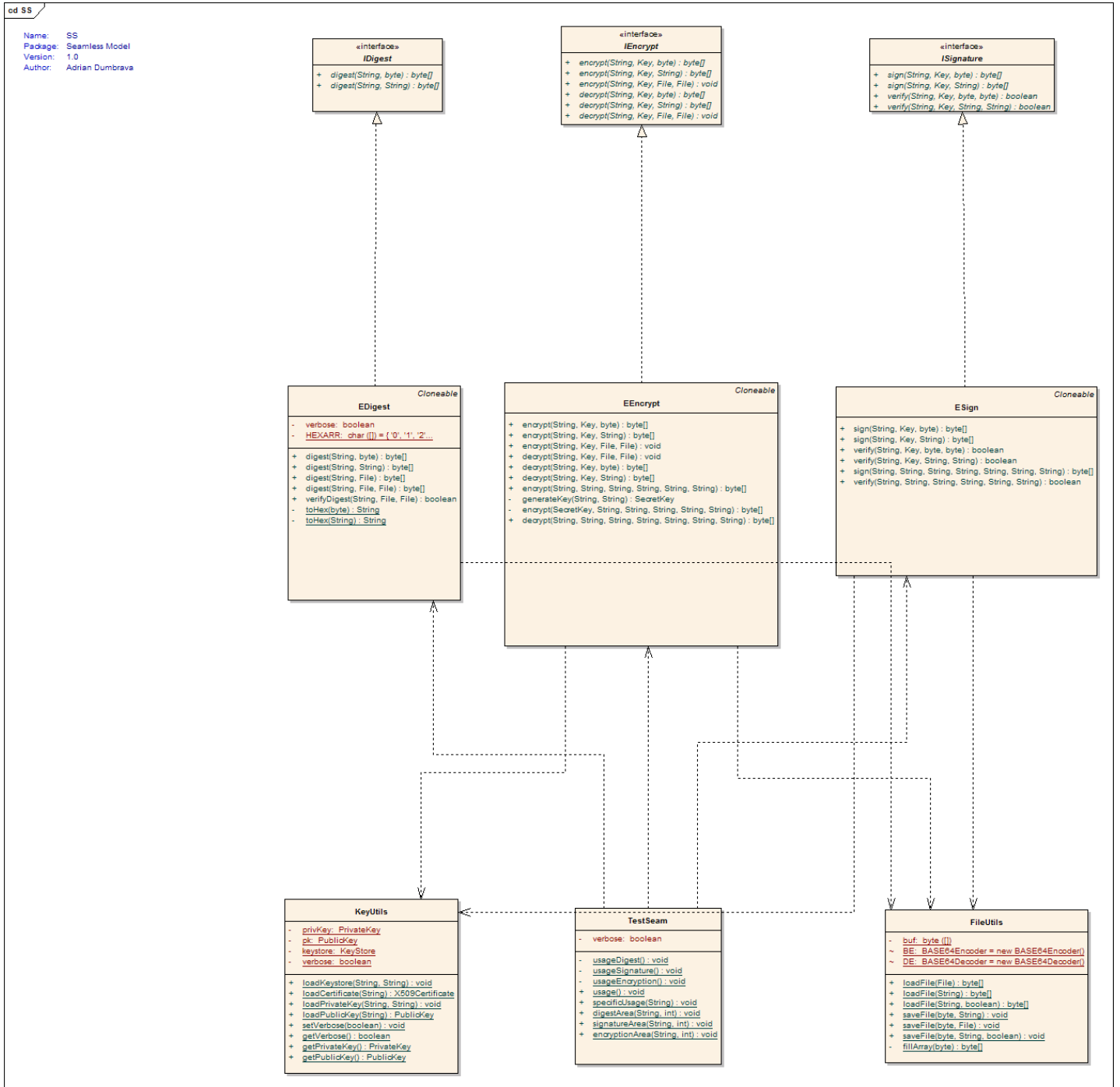
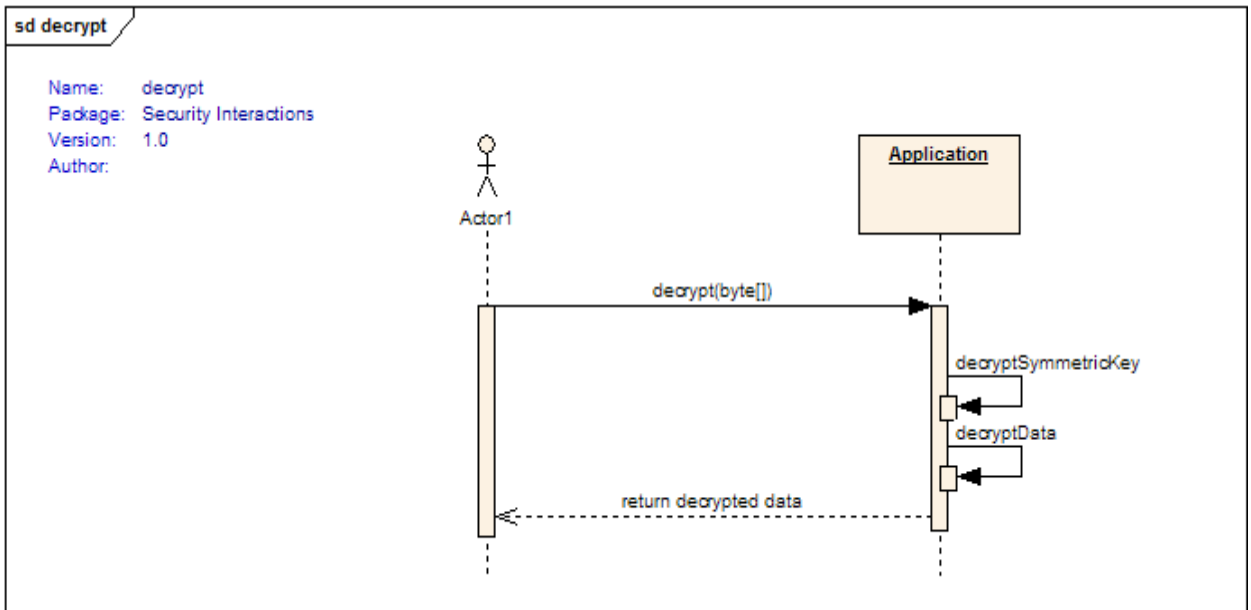
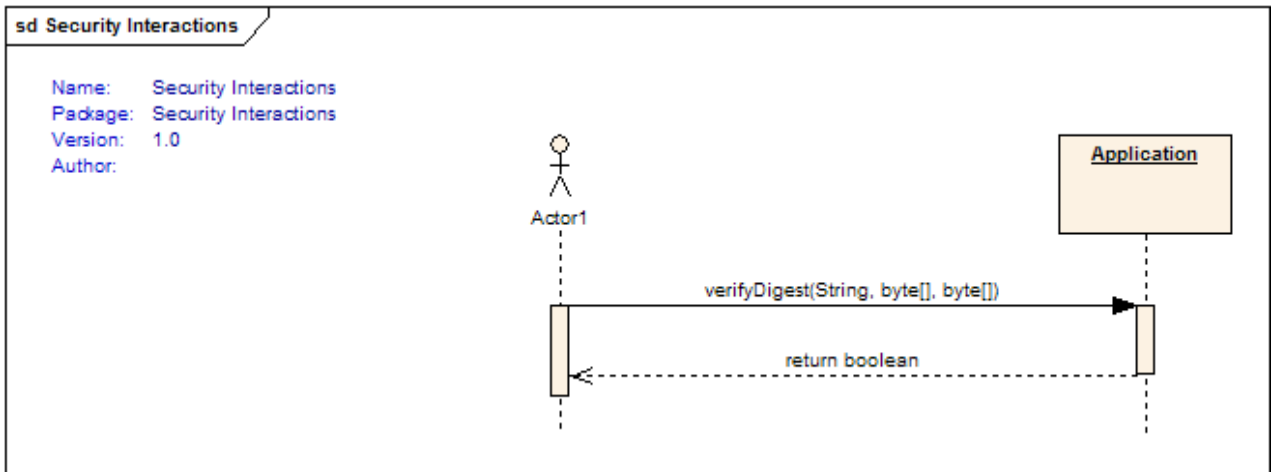
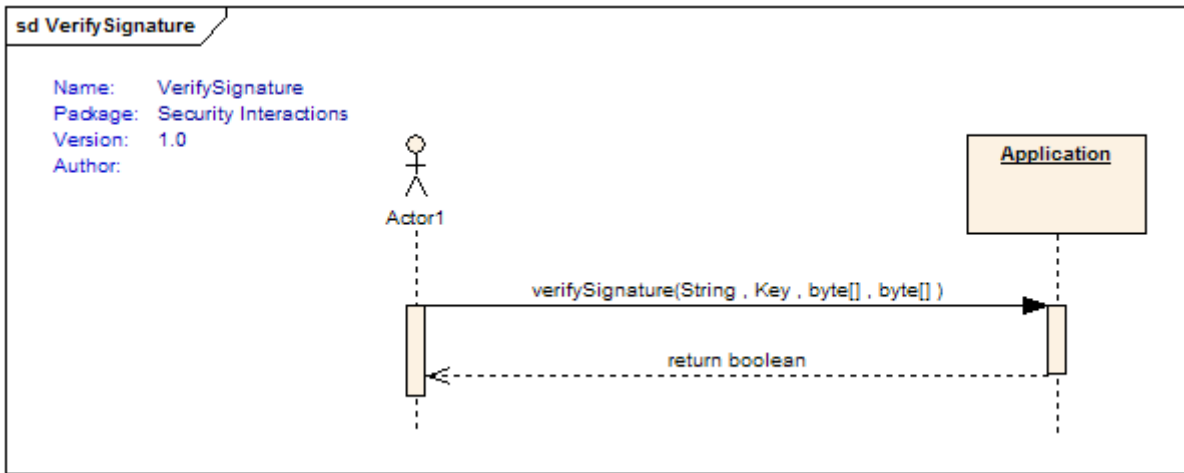


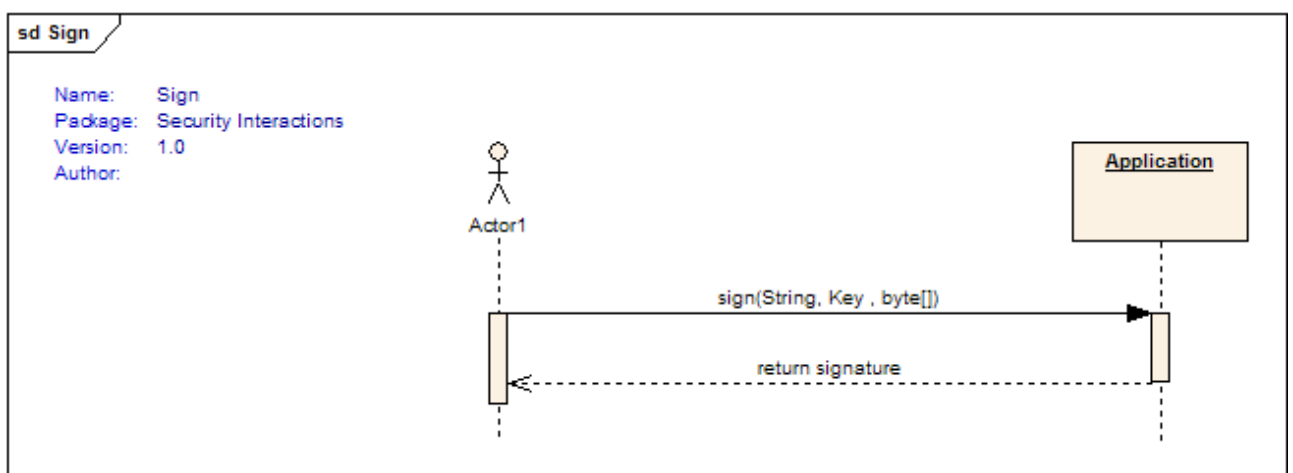
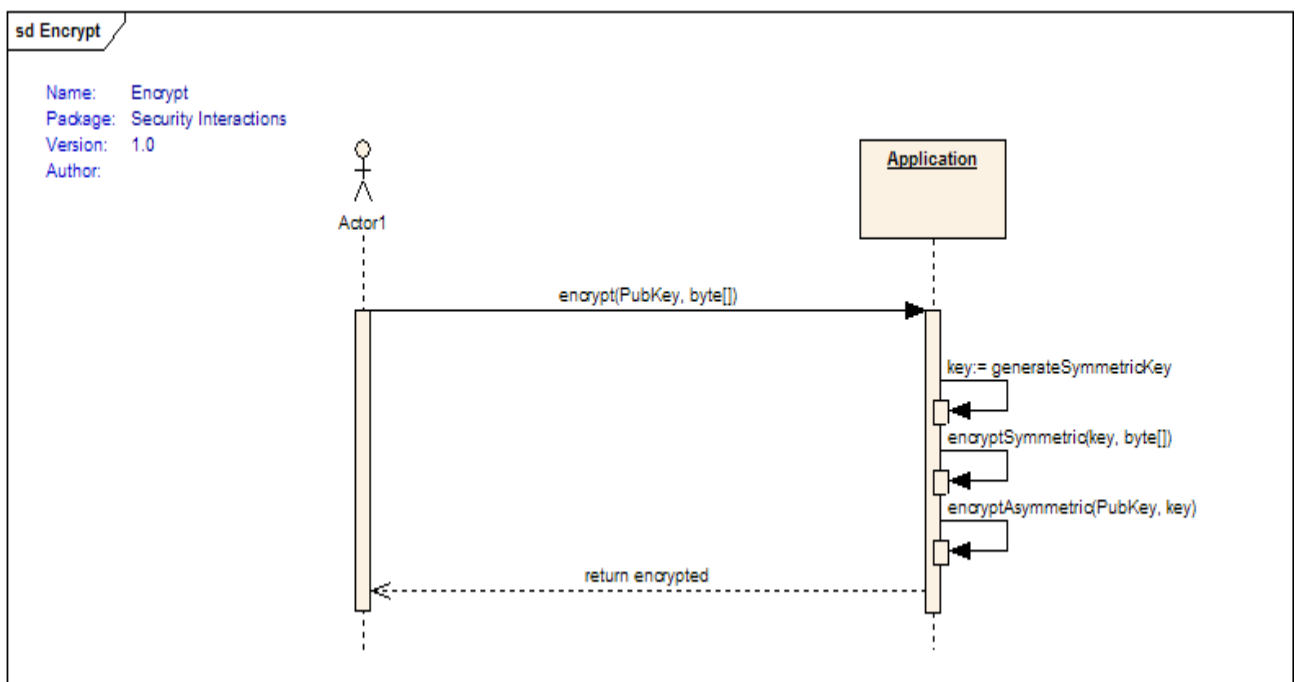
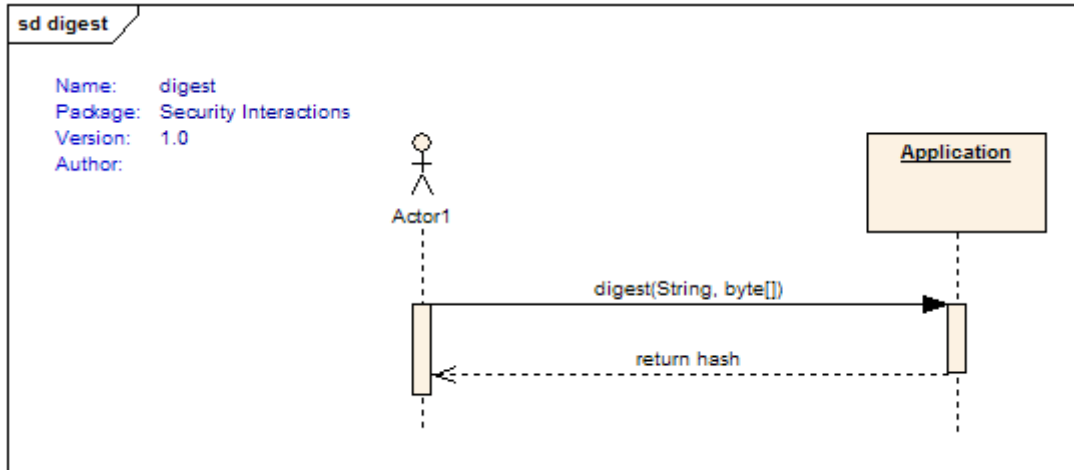
Figure 14: Data structure exchanged granting access to a certain object in the repository

ANNEX II – Class Diagram, Security Module



Annex III – Security Interactions Sequence Diagrams





References

- [1] Overall Architecture Design – SEAMLESS Project, WP 3.1 participants, July 21, 2006
- [2] SEAMLESS Description of Work, Annex I to EC Contract, September 5, 2005
- [3] OASIS Security Services http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=security
- [4] SAML 1.1 Specifications <http://www.oasis-open.org/specs/index.php#samlv1.1>
- [5] OpenSAML <http://www.opensaml.org/>
- [6] Liberty Alliance Project <http://www.projectliberty.org/>
- [7] Internet Shibboleth <http://shibboleth.internet2.edu/>
- [8] OASIS Web Services Security (WS-Security) <http://www.oasis-open.org/committees/wss/>
- [9] EJBCA (Certificate Authority) <http://ejbca.sourceforge.net/>
- [10] XML Trust Services (XKMS) <http://www.verisign.com/developer/xml/xkms.html>
- [11] LASSO - Free Liberty Alliance Implementation <http://lasso.entrouvert.org/>
- [12] WSS4J Apache's Documentation <http://ws.apache.org/wss4j/>
- [13] Axis Apache's Documentation <http://ws.apache.org/axis/>
- [14] SOA Modeling <http://www.soamodeling.net>

And mailing list archives <http://mail-archives.apache.org>

