



**Trusted Architecture for Securely Shared Services**

<b>Document Type:</b>	Deliverable
<b>Title:</b>	<b>TAS<sup>3</sup> Protocols, API, and Concrete Architecture</b>
<b>Work Package:</b>	WP2
<b>Deliverable Nr:</b>	D2.4
<b>Dissemination:</b>	Public
<b>Preparation Date:</b>	30 June 2010
<b>Version:</b>	13 (1.58)

**Legal Notice**

All information included in this document is subject to change without notice. The Members of the TAS3 Consortium make no warranty of any kind with regard to this document, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. The Members of the TAS3 Consortium shall not be held liable for errors contained herein or direct, indirect, special, incidental or consequential damages in connection with the furnishing, performance, or use of this material.

## The TAS<sup>3</sup> Consortium

	<b>Beneficiary Name</b>	<b>Country</b>	<b>Short</b>	<b>Role</b>
1	K.U. Leuven	BE	KUL	Project Mgr
2	Synergetics nv/sa	BE	SYN	Partner
3	University of Kent	UK	KENT	Partner
4	University of Karlsruhe	DE	KARL	Partner
5	Technische Universiteit Eindhoven	NL	TUE	Partner
6	CNR/ISTI	IT	CNR	Partner
7	University of Koblenz-Landau	DE	UNIKOLD	Partner
8	Vrije Universiteit Brussel	BE	VUB	Partner
9	University of Zaragoza	ES	UNIZAR	Partner
10	University of Nottingham	UK	NOT	Partner
11	SAP Research	DE	SAP	Coordinator
12	Eifel	FR	EIF	Partner
13	Intalio	UK	INT	Partner
14	Risaris	IR	RIS	Partner
15	Kenteq	BE	KETQ	Partner
16	Oracle	UK	ORACLE	Partner
17	Custodix	BE	CUS	Partner
19	Symlabs	PT	SYM	Partner

**Disclaimer:** This document has not been reviewed or approved by European Commission.

## Contributors

	<b>Name</b>	<b>Organisation</b>
1	Sampo Kellomäki (main contributor)	Unaffiliated
2	David Chadwick	KENT
3	Brecht Claerhout	CUS
4	Jeroen Hoppenbrouwers	KUL
5	Tom Kirkham	NOT
6	Brendan Van Alsenoy	KUL
7	Gang Zhao	VUB
8	Gilles Montagnon	SAP
9	Brian Reynolds	RIS

# Contents

<b>LIST OF FIGURES</b> .....	<b>8</b>
<b>EXECUTIVE SUMMARY</b> .....	<b>9</b>
<b>1 INTRODUCTION</b> .....	<b>10</b>
1.1 STANDARDIZED WIRE PROTOCOL INTERFACES.....	10
1.2 COMPOSITION AND CO-LOCATION OF ARCHITECTURAL COMPONENTS.....	11
<b>2 PROTOCOLS AND PROFILES</b> .....	<b>12</b>
2.1 SUPPORTED AUTHENTICATION AND LOGIN SYSTEMS.....	12
2.1.1 System Entity Authentication .....	12
2.1.2 SAML .....	12
2.1.3 Shibboleth.....	14
2.1.4 eID and Other Smart Cards .....	14
2.1.5 One-Time-Password Tokens .....	14
2.1.6 OpenID .....	14
2.1.7 CardSpace / InfoCard and WS-Federation.....	14
2.1.8 CA / Netegrity Siteminder Proprietary SSO .....	14
2.1.9 Citrix, Sun, and other proprietary SSO.....	15
2.1.10 Web Local Login .....	15
2.1.11 Desktop Login .....	15
2.1.12 Fat Client Login.....	15
2.1.13 User Not Present or Batch Operations .....	16
2.2 SUPPORTED IDENTITY WEB SERVICES SYSTEMS .....	16
2.2.1 Framework .....	16
2.2.2 Liberty ID-WSF Profile .....	17
2.2.3 Bare WS-Security Header or Simplified ID-WSF .....	18
2.2.4 WS-Trust.....	18
2.2.5 RESTful Approach .....	18
2.2.6 Message Bus Approach .....	19
2.3 AUTHORIZATION SYSTEMS .....	19
2.3.1 Authorization Queries.....	19
2.3.2 Policy Languages.....	19
2.4 TRUST AND SECURITY VOCABULARIES.....	19
2.4.1 Levels of Authentication (LoA).....	20
2.4.2 Vocabularies for Authorization .....	20
2.4.3 Vocabularies for Basic Attributes (PII).....	20
2.4.4 Discovery Vocabularies.....	20
2.4.5 Security and Trust Vocabularies .....	20

2.4.6	Audit Vocabularies .....	20
2.5	REALIZATION OF THE DISCOVERY FUNCTION .....	21
2.6	REALIZATION OF THE CREDENTIALS AND PRIVACY NEGOTIATOR FUNCTION .....	21
2.6.1	Discovery in Credentials and Privacy Negotiation .....	21
2.6.2	Frontend Credentials and Privacy Negotiation .....	22
2.6.3	Components of Credentials and Privacy Negotiator .....	22
2.6.4	Protocol between Service Requester and the Credentials and Privacy Negotiation Agent 24	
2.6.5	Protocol between Credentials and Privacy Negotiation Agent and Attribute Aggregator	24
2.6.6	Protocol between Credentials and Privacy Negotiation Agent and Service .....	24
2.7	USING TRUST SCORING IN DISCOVERY .....	24
2.8	REALIZATION OF THE AUDIT AND DASHBOARD FUNCTION .....	25
2.8.1	Audit Event Bus .....	25
2.8.2	Audit Event Ontology .....	25
2.8.3	Dashboard Function .....	25
2.8.4	User Interaction .....	25
2.9	REALIZATION OF DELEGATION FUNCTION .....	25
2.10	ATTRIBUTE AUTHORITIES .....	26
2.11	TAS <sup>3</sup> SIMPLE OBLIGATIONS LANGUAGE (SOL) .....	26
2.11.1	SOL1 Query String Attributes .....	27
2.11.2	Matching Pledges to Sticky Policies and Obligations .....	30
2.11.3	Passing Simple Obligations Dictionaries Around .....	31
2.12	REALIZATION OF STICKY POLICIES .....	32
2.13	PASSING ADDITIONAL CREDENTIALS IN WEB SERVICE CALL .....	33
2.14	UNIFORM APPLICATION STATUS AND ERROR REPORTING .....	34
2.14.1	TAS <sup>3</sup> Status Header .....	34
2.14.2	TAS <sup>3</sup> Status Codes .....	35
2.14.3	TAS <sup>3</sup> Control and Reporting Points .....	35
2.15	REGISTRATION OF BUSINESS PROCESS MODELS .....	35
<b>3</b>	<b>THE OFFICIAL TAS<sup>3</sup> API (NORMATIVE, BUT NON-EXCLUSIVE) .....</b>	<b>36</b>
3.1	LANGUAGE INDEPENDENT DESCRIPTION OF THE API .....	36
3.1.1	Single Sign On (SSO) Alternatives .....	36
3.1.2	SSO: <i>ret = tas3_sso(conf, qs, auto_flags)</i> .....	37
3.1.3	Authorization: <i>decision = tas3_az(conf, qs, ses)</i> .....	39
3.1.4	Web Service Call: <i>ret_soap = tas3_call(cf, ses, svctype, url, di_opt, az_cred, req_soap)</i>	40
3.1.5	Requester out: <i>req_decor_soap = tas3_wsc_prepare_call(cf, ses, svctype, az_cred, req_soap)</i> .....	42
3.1.6	Requester in: <i>status = tas3_wsc_valid_resp(cf, ses, az_cred, res_decor_soap)</i> .....	42
3.1.7	Responder in: <i>tgtnid = tas3_wsp_validate(cf, ses, az_cred, soap_req)</i> .....	42
3.1.8	Responder out: <i>soap = tas3_wsp_decorate(cf, ses, az_cred, soap_resp)</i> .....	43
3.1.9	Explicit Discovery: <i>epr = tas3_get_epr(cf, ses, svc, url, di_opt, act, n)</i> .....	43

3.1.10	url = <i>tas3_get_epr_url(cf, epr)</i> .....	44
3.1.11	entityid = <i>tas3_get_epr_entid(cf, epr)</i> .....	44
3.1.12	a7n = <i>tas3_get_epr_a7n(cf, epr)</i> .....	44
3.1.13	SOAP Fault and Status Generation and Inspection .....	44
3.2	JAVA BINDING .....	45
3.2.1	Interface and Initialization .....	45
3.2.2	Initialize: cf = <i>tas3.new_conf_to_cf(conf)</i> .....	46
3.2.3	New session: ses = <i>tas3.new_ses(cf)</i> .....	46
3.2.4	SSO: ret = <i>tas3.sso_cf_ses(cf, qs_len, qs, ses, null, auto_flags)</i> .....	46
3.2.5	Authorization: decision = <i>tas3.az_cf_ses(cf, qs, ses)</i> .....	46
3.2.6	WSC: resp_soap = <i>tas3.call(cf, ses, svctype, url, di_opt, az_cred, req_soap)</i> .....	47
3.2.7	WSP: tgtnid = <i>tas3.wsp_validate(cf, ses, az_cred, soap_req)</i> .....	47
3.2.8	WSP: soap = <i>tas3.wsp_decorate(cf, ses, az_cred, soap_resp)</i> .....	47
3.2.9	Explicit Discovery: epr = <i>tas3.get_epr(cf, ses, svc, url, di_opt, act, n)</i> .....	47
3.2.10	url = <i>tas3.get_epr_url(cf, epr)</i> .....	48
3.2.11	entityid = <i>tas3.get_epr_entid(cf, epr)</i> .....	48
3.2.12	a7n = <i>tas3.get_epr_a7n(cf, epr)</i> .....	48
3.2.13	Available Implementations (Non-normative) .....	48
3.3	PHP BINDING .....	49
3.3.1	Application Level Integration .....	49
3.3.2	cf = <i>tas3_new_conf_to_cf(conf)</i> .....	49
3.3.3	ses = <i>tas3_new_ses(cf)</i> .....	49
3.3.4	SSO: ret = <i>tas3_sso_cf_ses(cf, -1, qs, ses, null, auto_flags)</i> .....	49
3.3.5	Authorization: decision = <i>tas3_az_cf_ses(cf, qs, ses)</i> .....	50
3.3.6	WSC: resp_soap = <i>tas3_call(cf, ses, svctype, url, di_opt, az_cred, req_soap)</i> .....	50
3.3.7	WSP: tgtnid = <i>tas3_wsp_validate(cf, ses, az_cred, soap_req)</i> .....	51
3.3.8	WSP: soap = <i>tas3_wsp_decorate(cf, ses, az_cred, soap_resp)</i> .....	51
3.3.9	Explicit Discovery: epr = <i>tas3_get_epr(cf, ses, svc, url, di_opt, act, n)</i> .....	51
3.3.10	url = <i>tas3_get_epr_url(cf, epr)</i> .....	51
3.3.11	entityid = <i>tas3_get_epr_entid(cf, epr)</i> .....	52
3.3.12	a7n = <i>tas3_get_epr_a7n(cf, epr)</i> .....	52
3.3.13	Available Implementations (Non-normative) .....	52
3.4	C AND C++ BINDING .....	53
3.4.1	cf = <i>tas3_new_conf_to_cf(conf)</i> .....	53
3.4.2	ses = <i>tas3_new_ses(cf)</i> .....	53
3.4.3	SSO: ret = <i>tas3_sso_cf_ses(cf, qs_len, qs, ses, &amp;res_len, auto_flags)</i> .....	53
3.4.4	Authorization: decision = <i>tas3_az_cf_ses(cf, qs, ses)</i> .....	54
3.4.5	WSC: resp_soap = <i>tas3_call(cf, ses, svctype, url, di_opt, az_cred, req_soap)</i> .....	54
3.4.6	resp_soap = <i>tas3_callf(cf, ses, svctype, url, di_opt, az_cred, fmt, ...)</i> .....	55
3.4.7	WSP: tgtnid = <i>tas3_wsp_validate(cf, ses, az_cred, soap_req)</i> .....	55
3.4.8	WSP: soap = <i>tas3_wsp_decorate(cf, ses, az_cred, soap_resp)</i> .....	56

- 3.4.9 WSP: soap = *tas3\_wsp\_decoratef(cf, ses, az\_cred, fmt, ...)* ..... 56
- 3.4.10 Explicit Discovery: *epr = tas3\_get\_epr(cf, ses, svc, url, di\_opt, act, n)* ..... 56
- 3.4.11 *url = tas3\_get\_epr\_url(cf, epr)* ..... 57
- 3.4.12 *entityid = tas3\_get\_epr\_entid(cf, epr)* ..... 57
- 3.4.13 *a7n = tas3\_get\_epr\_a7n(cf, epr)* ..... 57
- 3.4.14 Available Implementations (Non-normative) ..... 57
- 3.5 OTHER LANGUAGE BINDINGS ..... 58
- 4 DEPLOYMENT AND INTEGRATION MODELS (NON-NORMATIVE)..... 59**
- 4.1 FRONTEND AND WEB SERVICES CLIENT INTEGRATION MODEL (NON-NORMATIVE)... 59
  - 4.1.1 Integration Using ZXID (Non-normative) ..... 60
  - 4.1.2 Integration Using Other Platforms, Frameworks, and Packages (Non-normative) ..... 62
- 4.2 WEB SERVICES PROVIDER INTEGRATION MODEL (NON-NORMATIVE)..... 62
- 5 RESILIENT DEPLOYMENT ARCHITECTURE (NON-NORMATIVE) ..... 63**
- 5.1 ZERO DOWNTIME UPDATES ..... 64
- 6 FEASIBILITY AND PERFORMANCE ANALYSIS (NON-NORMATIVE)..... 65**
- 6.1 SINGLE USE OF SINGLE WEB SERVICE ..... 66
  - 6.1.1 Cost without auditing ..... 67
  - 6.1.2 Cost without auditing and without authorization ..... 67
  - 6.1.3 Cost without XML..... 68
- 6.2 SESSION OF 3 FRONTENDS AND FIVE WEB SERVICES ..... 68
- 7 ANNEX A: EXAMPLES ..... 71**
- 7.1 SAML 2.0 ARTIFACT RESPONSE WITH SAML 2.0 SSO ASSERTION AND TWO BOOTSTRAPS ..... 71
- 7.2 ID-WSF 2.0 CALL WITH X509v3 SEC MECH ..... 74
- 7.3 ID-WSF 2.0 CALL WITH BEARER (BINARY) SEC MECH ..... 75
- 7.4 ID-WSF 2.0 CALL WITH BEARER (SAML) SEC MECH ..... 76
- 8 ANNEX B: TECHNICAL SELF ASSESSMENT QUESTIONNAIRE..... 79**
- 8.1 OVERVIEW AND SCOPE..... 79
- 8.2 SYSTEM ENTITY CREDENTIALS AND PRIVATE KEYS..... 80
- 8.3 TRUST MANAGEMENT ..... 81
- 8.4 THREAT AND RISK ASSESSMENTS..... 82
- 8.5 SERVICE PROVIDER QUESTIONS ..... 82
  - 8.5.1 Front End (FE) Single Sign-On Questions ..... 82
  - 8.5.2 Web Service Provider (WSP) Questions ..... 83
  - 8.5.3 Attribute Authority Questions ..... 84
  - 8.5.4 Web Service Client (WSC) Questions ..... 85

8.6 SINGLE SIGN-ON IDENTITY PROVIDER (IDP), DISCOVERY SERVICE, DISCOVERY REG-  
ISTRY, IDENTITY MAPPER, OR DELEGATION SERVICE QUESTIONS ..... 86

    8.6.1 Identity Provider Questions ..... 86

    8.6.2 Discovery Service Questions ..... 87

8.7 ANY OTHER ARCHITECTURAL ROLE ..... 87

**BIBLIOGRAPHY ..... 88**

# List of Figures

Figure 2.1: Liberty Alliance Architecture. . . . . 18

Figure 2.2: Hierarchy of policies . . . . . 20

Figure 2.3: Credentials and Privacy Negotiation and Discovery steps. . . . . 21

Figure 2.4: A deployment architecture for Credentials and Privacy Negotiation and Discovery. . . . . 22

Figure 2.5: Credentials and Privacy Negotiation Components. . . . . 23

Figure 2.6: Credentials and Privacy Negotiation optimized flow . . . . . 23

Figure 4.1: A deployment architecture for SSO and web service call. . . . . 59

Figure 4.2: API and modules for SSO and web service call. . . . . 61

Figure 4.3: ZXID specific API and modules for SSO and web service call. . . . . 61

Figure 5.1: Layering of resilience features for Front Channel, Back Channel, and data center Back End services. . . . . 63

Figure 5.2: Resiliency implemented using hardware load balancers. . . . . 63

Figure 5.3: Resiliency implemented using software load-balancing-fail-over functionality and clustering. . . . . 64

## Keyword List

<sup>1</sup> Architecture, Protocol, Implementation, API, Security, Trust, Privacy

2

## 3 Protocols and Concrete Architecture Executive Summary

4 This document specifies a set of protocol level interoperability profiles, usually leveraging open stan-  
5 dards, deployment scenarios, APIs, and other considerations that constitute the official way to deploy  
6 version 1 of TAS<sup>3</sup> architecture, see [[TAS3ARCH](#)]. The purpose of defining these specifics is to enable  
7 multiple independent implementations of TAS<sup>3</sup> to be wire protocol interoperable (and to limited extent  
8 also API interoperable). TAS<sup>3</sup> reference implementation and reference deployment will behave essen-  
9 tially as described in this document.

10 The TAS<sup>3</sup> architecture is designed to be standards, protocol, data and application agnostic so that any  
11 protocol capable of implementing the flows and satisfying the service requirements can potentially be  
12 used by any application. However, to build practical systems, different components, possibly from differ-  
13 ent sources, must speak the same protocols, hence TAS<sup>3</sup> provides this profile that allows interoperability at  
14 the level of Single Sign-On, Web Service Discovery, Web Service Call, and Authorization. The standard-  
15 ized profile provides the scaffolding where plurality of trust and privacy negotiation mechanisms, policy  
16 languages, obligations and other value added features can exist.

17 The TAS<sup>3</sup> API is designed to allow an application programmer to understand how simple it is to "TAS<sup>3</sup>  
18 enable" his application. It is noteworthy that using the API does not require any in-depth knowledge of  
19 the underlying standards, protocols, and profiles, or indeed even of the TAS<sup>3</sup> Architecture itself. All these  
20 details are taken care of by the API implementation, supplied commercially or in open source. The TAS<sup>3</sup>  
21 Reference Implementation will be one such API implementation. The APIs will be available in all popular  
22 programming languages and platforms.

23 The simplicity of the API is due to a coherent integration model that shows how the steps from SSO and  
24 Authorization all the way to the web service calls work together and are able to pass necessary credentials  
25 and tokens "behind the scenes" by the use of session and other state information. Many design parameters  
26 that could have been handled by yet another argument to the API functions, are in fact handled by con-  
27 figuration file, with sensible default values, and automated discovery, trust negotiation, and trust network  
28 business processes.

29 The split between explicit arguments, configurability, and automated processes has been guided by  
30 division of concerns between the application programmer and the systems administrator. When automatic  
31 mechanisms are used, appropriate manual control point exists elsewhere in the architecture, e.g. automated  
32 discovery is kept in check with explicit authorization.

33 We provide guidance regarding possible integration and deployment scenarios and illustrate how TAS<sup>3</sup>  
34 Architecture can be deployed in a resilient and redundant way.

35 Neither this document nor the TAS<sup>3</sup> Architecture [[TAS3ARCH](#)] mandate use of a particular deployment  
36 or software architecture (although the integration scenarios suggest a recommended one), implementers  
37 are free to organize their software and deployment in other ways as long as the wire protocol compati-  
38 bility is maintained and all signature generation and validation steps, as well as trust determinations, and  
39 authorizations are implemented.

40 The Annex gives some example protocol messages.

# 1 Introduction

This document describes the TAS<sup>3</sup> Concrete Architecture and protocol choices in a normative and prescriptive way. It also describes the official, but not exclusive, TAS<sup>3</sup> API generically and for selected programming language bindings. Any implementation or deployment claiming "TAS<sup>3</sup>" compliance MUST abide by this document as well as [TAS3ARCH], and [TAS3COMPLIANCE]. A deployment usually has to satisfy, as well, requirements of the Trust Operator's, see [TAS3GLOS], Governance Agreement and certification procedures, some of which concern the software implementation and others the deployment's organizational properties. Use of TAS<sup>3</sup> brand is governed by a separate TAS<sup>3</sup> Brand Agreement.

This document uses the keywords (e.g. MUST, SHOULD) of [RFC2119]. All text is normative unless expressly identified as non-normative. Prose and specification has precedence over examples. In general the examples should not be assumed normative unless no normative specification for the subject matter is available.

This architecture, and related documents are copyrighted works of TAS3 Consortium, as dated. All Rights Reserved. This architecture, and related documents, are versioned and subject to change without notice. No warranty or guarantee is given. This architecture, and related specifications can be implemented on Royalty Free terms by anyone. However, no warranty regarding IPR infringement is given. For further details, please see [TAS3CONSOAGMT].

## 1.1 Standardized Wire Protocol Interfaces

TAS<sup>3</sup> emphasizes wire protocol interoperability in following key areas

1. Single Sign-On (SSO) and Single Logout (SLO)
2. Authorization request-response
3. ID Mapping and Discovery
4. Web service call
5. Audit bus reporting and audit trail querying
6. Delegation
7. Metadata, registrations, declarations of attribute needs, declarations of attribute availability

In some areas TAS<sup>3</sup> recognizes interoperability need, but leaves it up to the business processes, adaptive techniques, and involved parties to agree specific means. These include

- Policy and obligations languages and vocabularies (although we suggest XACML and SOL1, see section 2.11, as one alternative, supported by the reference implementation)
- Trust and Privacy Negotiation protocol and metrics or scores (although we suggest TrustBuilder and some XACML extensions, see section ??)
- Application ("payload") protocols and data formats
- Format of the local audit trail
- Business Process Modelling techniques and languages

TAS<sup>3</sup> recognizes the usefulness of a consistent user experience, e.g. in Dashboard, SSO, consent, trust and privacy negotiation, policy editing, etc., but this document does not attempt to prescribe these aspects.

80

## 81 1.2 Composition and Co-location of Architectural Components

82 This section addresses Req. *D1.2-3.8-Separate*, *D1.2-2.24-NoPanopt*, *D1.2-6.80-Separate*.

83 When implementing practical systems, it often turns out that many of the architecturally designed  
84 boxes are in fact implementable by one software module. For example, with reference to Fig-2.3 of  
85 [TAS3ARCH], it is clear that a software module called "Service Requester" may exist, realizing Rq-  
86 PEP-Out, Rq-PEP-In, and Stack components all together without them being necessarily separable. Such  
87 composition does not harm interoperability as those submodules of Service Requester were always meant  
88 to be part of the same process and to communicate via function call interfaces. Indeed, the official TAS<sup>3</sup>  
89 API, see section 3, lumps all these in one function call: *tas3\_call()*. However all external interfaces from  
90 *tas3\_call()*, such as authorization, discovery, and web service call, do speak standard protocols as profiled  
91 in this document.

92 It is ok for an implementation to compose, as an optimization, components that were meant to be wire  
93 protocol interfaces (see section 1.1), e.g. reach authorization by function call interface instead of XACML,  
94 as long as the implementation makes the same interface available over-the-wire by a mere configuration  
95 change (no recompile required/allowed).

96 From protocol perspective *co-location* of services (having two distinct service processes running on the  
97 same server hardware, or even running as separate processes under the same web server) does not present  
98 any problem, save for the complications of using nonstandard TCP/IP ports or requirement of configuring  
99 multiple IP addresses to same host.

100 From risk management and excessive visibility, or fat target, perspective, see *T161-Panopticon* threat  
101 in [TAS3COMPLIANCE], some services clearly should not be co-located. Division of responsibilities  
102 becomes important here and any two roles played by one system entity where they are co-located must  
103 not have a conflict of interest. In particular, the following are incompatible for co-location

- 104 ● anything vs. Audit
- 105 ● SP vs. IdP (some exceptions apply)
- 106 ● SP vs. ID Mapping and Discovery
- 107 ● SP vs. Delegation
- 108 ● IdP vs. Authorization (some exceptions apply)

109 Some services can be safely co-located, and often are:

- 110 ● IdP often includes Attribute Authority, ID Mapping, Discovery, and fat client Authentication Ser-  
111 vice. Although an IdP should not pretend to be a Policy Enforcement Point, it is clear that an IdP  
112 can exert such control by refusing to issue tokens that are necessary for functioning of the rest of  
113 the architecture.
- 114 ● SP and PEP are natural partners, indeed different facets of the same process

115

116

## 2 Protocols and Profiles

117

To complement the specification of protocols here, the reader may want to consult Fig-8.18 in [HafnerBreu09] for an overview of the functionality available in various specifications.

119

120

122

The choice of protocols has been guided by commitment to open standards as recommended in section 2 of [UNDP07]. This also serves to address Reqs. *D1.2-2.4-MultiVendor*, *D1.2-2.5-Platform*, and *D1.2-2.6-Lang*.

123

### 2.1 Supported Authentication and Login Systems

124

126

This section addresses Reqs. *D1.2-2.18-AnCredi*, *D1.2-6.12-Sec*, *D1.2-7.3-An*, *D1.2-7.10-Target*, *D1.2-9.3-SSO*.

127

#### 2.1.1 System Entity Authentication

128

129

TAS<sup>3</sup> adopts X.509v3 public key certificates as primary means of authenticating system entities. This will apply over TLS and ClientTLS connections and may also apply in digital signatures.

130

132

For bilateral authentication Client TLS MUST be supported. HTTP Basic authentication MAY be supported.

133

#### 2.1.2 SAML

134

135

136

137

Given the already broad adoption of SAML 2.0 by the eGovernment and academic communities across the world (e.g. DK, NZ, FI, etc.), this choice is effectively already made for us. By choosing SAML 2.0 we enable many existing eGovernment and academic projects easily to become TAS<sup>3</sup> compliant in future.

138

139

140

141

142

1. TAS<sup>3</sup> adopts SAML 2.0 Assertions, see [SAML2core], as primary and recommended token format. Alternatives such as SAML 1.1 or Simple Web Token (SWT) [Hardt09] were considered either obsolete or not yet mature. In future we may consider supporting SWT and X509 attribute certificates as token format. This will become especially relevant when architecture is extended to support RESTful services approaches.

143

144

2. TAS<sup>3</sup> adopts SAML 2.0 as primary and RECOMMENDED SSO system, see [SAML2core]. (Req. *D1.2-3.10-JITPerm*)

145

3. TAS<sup>3</sup> RECOMMENDS that SAML 2.0 implementations are Liberty Alliance Certified.

146

4. SAML 1.0, 1.1 [SAML1core], 1.2, as well as Liberty ID-FF 1.2 [IDFF12] MAY be supported

147

148

5. Redirect - POST SSO profile MUST be supported by all front channel participants, see [SAML2prof] and [SAML2bind].

149

150

6. Redirect - Artifact - SOAP SSO profile MUST be supported in IdP and SHOULD be supported in Front End (SP), see [SAML2prof] and [SAML2bind].

151

7. Redirect Single Logout Profile MUST be supported, see [SAML2prof] and [SAML2bind].

152

8. IdP Extended Profile, see [SAML2conf], namely IdP Proxying, MUST be supported

153

9. Other SAML profiles MAY be supported

154

10. SAML metadata MUST be supported, see [SAML2meta]

155

156

157

11. Well Known Location (WKL) method of metadata publishing MUST be supported, see [SAML2meta] section 4.1 "Publication and Resolution via Well-Known Location", p.29, for normative description of this method. Support for WKL method for metadata acquisition is RECOMMENDED.

158 N.B. Publishing metadata using WKL at its most basic form is as simple as placing a hand  
 159 edited metadata file in the web root at the place referenced by the EntityID of the site.  
 160 Many software packages handle this automatically and may even generate the metadata  
 161 dynamically, on the fly.

162 12. In redirect binding [[RFC1951](#)] deflate compression MUST be used. [[RFC1952](#)] format MUST NOT  
 163 be used.  
 164

165 **2.1.2.1 Authentication Request**

- 166 1. MUST use `NameIDPolicy/@Format` of Persistent ("urn:oasis:names:tc:SAML:2.0:nameid-format:persistent")  
 167 when implementing Pull Model (Req. *DI.2-7.8-NoColl*).
- 168 2. MUST use `NameIDPolicy/@Format` of Transient ("urn:oasis:names:tc:SAML:2.0:nameid-format:transient")  
 169 when implementing Linking Service model.
- 170 3. MUST set `NameIDPolicy/@SPNameQualifier`
- 171 4. MUST set `NameIDPolicy/@AllowCreate` flag at all times true
- 172 5. SHOULD not set `IsPassive` flag (in some cases there may be justified reasons to do otherwise)
- 173 6. MUST use `AssertionConsumerServiceIndex`
- 174 7. MUST NOT use `ProtocolBinding` or `AssertionConsumerServiceURL`
- 175 8. Step-up authentication, using Authentication Context Class References MUST be supported.
- 176 9. SHOULD use `AttributeConsumingServiceIndex` attribute, which refers to a section of the meta-  
 177 data, as way of selecting the attributes that are returned in the authentication response. Reader should  
 178 be aware that new proposals for solving this issue more dynamically have been submitted to OASIS  
 179 Security Services Technical Committee, e.g. [[Kellomaki08](#)]. It should also be noted that the returned  
 180 attributes are always at discretion of the IdP.  
 181

182 **2.1.2.2 Authentication Response**

183 The authentication request will be responded with an assertion that satisfies following:

- 184 1. MUST contain `<sa:AuthnStatement>`
- 185 2. MUST specify the Level of Authentication as `AuthnStatement/AuthnContext/AuthnContextClassRef`.
- 186 3. MUST use the LoA profile [[SAML2LOA](#)] to return LoA to the SP.
- 187 4. SHOULD have `AudienceRestriction/Audience` element referencing the SP.
- 188 5. MAY contain `<AttributeStatement>` detailing user's attributes as relevant to SP and/or requested  
 189 using `AttributeConsumingServiceIndex`.
- 190 6. SHOULD have an `<AttributeStatement>` containing a discovery bootstrap (attribute named "urn:liberty:disco:2006-  
 191 08:DiscoveryEPR" whose value is an endpoint reference) as described in [[Disco2](#)] section 4 "Discovery  
 192 Service ID-WSF EPR conveyed via a Security Token".
- 193 7. MAY have additional Attribute Statements conveying other endpoint references. Rather than providing  
 194 additional EPRs at SSO, using discovery is RECOMMENDED. If additional EPRs are passed, the  
 195 attributes SHOULD be named "urn:liberty:disco:2006-08:DiscoveryEPR" even if they do not refer  
 196 to discovery service. The SP, when seeing "urn:liberty:disco:2006-08:DiscoveryEPR" attribute MUST  
 197 look at the `Attribute/AttributeValue/EndpointReference/Metadata/ServiceType` element to  
 198 determine the type of the end point reference. The SP SHOULD consider any attribute whose value is  
 199 an `<a:EndpointReference>` to be a bootstrap.

200

### 201 **2.1.3 Shibboleth**

202 Shibboleth MAY be supported. Shibboleth based on SAML 2.0 is RECOMMENDED. Supporting  
203 Shibboleth enables higher education institutions to adopt TAS<sup>3</sup> with minimal reconfiguration and rein-  
204 vestment.

205 Shibboleth does not currently (2009) support Single Logout. As a condition of TAS<sup>3</sup> compliance, such  
206 support should be added (please contribute any such work to the Shibboleth open source implementation  
207 so that this caveat can be deleted). However, a TAS<sup>3</sup> compliant Trust Network may waive this requirement  
208 after analysis of the impact and a pondered decision (i.e. its easier to implement it than to get lawyers to  
209 agree).

210 Shibboleth does not officially support Well Known Location method of metadata publication, but any  
211 Shibboleth deployment can satisfy this requirement by simply hand crafting a metadata file and making it  
212 available on their web server at the EntityID URL.

213 We have not fully validated all use cases with Shibboleth. Specific points of contention include lack of  
214 full user identification, e.g. statement that User is a student or staff member of university, without giving  
215 out a persistent pseudonym. While a valid approach that better protects the user's privacy than the use of  
216 a persistent ID, it may not be able to address all the use cases, especially in the commercial world where  
217 service providers wish to link a user's requests together.

### 219 **2.1.4 eID and Other Smart Cards**

220 European eID cards and other smart cards are supported as an authentication method available at SAML  
221 2.0 IdP.

### 223 **2.1.5 One-Time-Password Tokens**

224 One-Time-Password Tokens, such as RSA Tokens or Yubikey, are supported as an authentication meth-  
225 ods available at SAML 2.0 IdP.

### 227 **2.1.6 OpenID**

228 OpenID [[OpenID](#)] MAY be supported. If supported, OpenID 2.0 MUST be used as earlier versions  
229 have known security flaws.

230 It should be noted that OpenID's globally unique identifier model does not provide privacy protection.

231 We have not validated whether it is possible to implement TAS<sup>3</sup> architecture using OpenID. One specific  
232 point of uncertainty is passing the IM bootstrap token at SSO time. No native OpenID mechanism is  
233 known to exist (standardized; ad-hoc approaches are known). One suggestion, applicable to the RESTful  
234 binding would be to use OAUTH.

### 236 **2.1.7 CardSpace / InfoCard and WS-Federation**

237 Card Space MAY be supported. If supported, at least SAML 2.0 token format MUST be supported.  
238 The token MUST also support passing IM / Discovery bootstrap token.

### 240 **2.1.8 CA / Netegrity Siteminder Proprietary SSO**

241 Siteminder MAY be supported. However, we have not validated whether it is possible to implement  
242 TAS<sup>3</sup> architecture using Siteminder. Prospects do not look particularly good as the Siteminder protocol  
243 and product can not easily be configured to convey the IM bootstrap token. However, the same vendor  
244 sells a SAML2 solution, so ask for that instead.

- 245 • Not standards compliant, but by far the most relevant player on the market

246

### 247 **2.1.9 Citrix, Sun, and other proprietary SSO**

248 MAY be supported. However, we have not validated whether it is possible to implement TAS<sup>3</sup> archi-  
249 tecture using these.

### 251 **2.1.10 Web Local Login**

252 We have not validated whether it is possible to implement TAS<sup>3</sup> architecture using local login approach.  
253 The local login approach has many problems, including

- 254 • Each site has separate login so more burden to the user
- 255 • Users are lazy and use same password on many sites, thus allowing the sites to impersonate (mas-  
256 querade) their users towards other sites.
- 257 • Local logins require local effort to support new better authentication methods.
- 258 • Local logins necessitate local user database maintenance
- 259 • Local logins require password resets to be handled locally

260 If you must do local login, we recommend using one-time-passwords and the Authentication Service  
261 Protocol [[SOAPAuthn2](#)] to validate the authentication centrally using an IdP.

### 263 **2.1.11 Desktop Login**

264 We have not validated whether it is possible to implement TAS<sup>3</sup> architecture using desktop login ap-  
265 proach. We recommend using one-time-passwords and the Authentication Service Protocol [[SOAPAuthn2](#)]  
266 to validate the authentication centrally using an IdP.

- 267 • Terminal servers: Mind-The-Box, Citrix, Windows TS, etc.
- 268 • Active Directory PDC

269 A backup plan would be to capture the authentication at LDAP or Active Directory level and make the  
270 Authentication Service call from this middleware.

271 The Desktop login approach suffers from similar security problems as the Fat Client Login, which see  
272 below.

### 274 **2.1.12 Fat Client Login**

275 "Fat Client" refers to any non web browser client, e.g. email reading program (as opposed to web mail)  
276 or GUI form filling application (as opposed to web GUI). Fat Client scenario often arises with embedded  
277 systems, such as medical devices that need to talk to TAS<sup>3</sup> network.

278 The main security problem in Fat Client Login is that the fat client itself becomes an intermediary to  
279 the authentication process, handling sensitive credentials. Some notion of Trusted Computing Path may  
280 help to address verifying that the fat client is not compromised.

281 We recommend using one-time-passwords and the Authentication Service Protocol [[SOAPAuthn2](#)] to  
282 validate the authentication centrally using an IdP. One-time-passwords effectively solve the intermediary  
283 problem.

284 If Fat Client Login is a requirement, Liberty Advanced Client approach, see [[AdvClient](#)] and [[SOAPAuthn2](#)],  
285 SHOULD be used.

286

### 287 2.1.13 User Not Present or Batch Operations

288 TAS<sup>3</sup> specifies some approaches for doing this, see [TAS3D4IID], mainly based on using advanced au-  
 289 thorization to obtain discovery token without authenticating the User. Liberty Advanced Client approach,  
 290 see [AdvClient] and [SOAPAuthn2], SHOULD be used.

## 292 2.2 Supported Identity Web Services Systems

293 The web services must satisfy some technical requirements

- 294 ● Messages MUST be correlated, so each response is bound to request in an auditable way
  - 295 - Message ID correlation
  - 296 - Business Process Model and Instance IDs (or context or instance) to allow overarching correla-  
 297 tion of several request-response pairs (e.g. to avoid actors who would have conflicts of interest  
 298 overall that might not be identified when only working at level of individual request-response  
 299 pairs)
    - 300 - PDP can receive this easy enough as an environment parameter and this is needed to  
 301 support dynamic separation of duties
    - 302 - Gap: business process modelling does not express this?
    - 303 - Consider URL format hierarchical ID
    - 304 - Better typed, like LDAP DN format, or query string
- 305 ● Requester and Responder MUST be identified (Req 10.4)
- 306 ● Synchronous web service calls MUST be supported
- 307 ● Asynchronous calls SHOULD be supported where needed. Business Process Engines will handle  
 308 asynchrony.
- 309 ● Subscribe - Notify mechanism SHOULD be supported where needed
  - 310 - subscription for events will be vital to pick up errors and notify of events like break the glass
  - 311 - subscribe and publish ws-eventing
  - 312 - Event bus as a subscribe and publish mechanism
- 313 ● Maximum availability and use digital signature and encryption technologies, i.e. technical solutions  
 314 to security and trust problems.

315

### 316 2.2.1 Framework

- 317 1. MUST support SOAP 1.2
- 318 2. MUST support XML-DSIG [XMLDSIG], a.k.a. RFC3275. In future we may introduce simpler  
 319 schemes like Simple Web Token [Hardt09]. Using TLS connection stream as an audit trail element  
 320 is impractical due to volume and inability of implementations to capture it. TLS stream as audit trail  
 321 may also lead to inadvertent collateral disclosure.
- 322 3. MUST support Exclusive XML Canonicalization [XML-EXC-C14N] for purposed of [XMLDSIG].
- 323 4. MAY support simple sign [SAML2SimpleSign]. In future we will support Simple Web Token [Hardt09]  
 324 which is very similar to simple sign.

325 5. MUST support XML-Enc [XMLENC] for protection of NameIDs and attributes, including bootstraps,  
 326 as well as assertions, against an active intermediary. The common case in question is a SP that is about  
 327 to make a web service call. To make such call, the SP must obtain from the discovery service a token  
 328 that is passed to the web service provider. XML-Enc support allows the discovery service to pass in the  
 329 encrypted token the pseudonym, and potentially some sensitive attributes, to the web service provider  
 330 without the intermediary, SP in this case, being able to snoop on this confidential information. This  
 331 case can not be solved using TLS alone as TLS is point-to-point and for this case TAS<sup>3</sup> architecture  
 332 necessarily specifies an active intermediary.  
 333

### 334 2.2.2 Liberty ID-WSF Profile

335 1. MUST support ID-WSF 2.0 SOAP Binding [SOAPBinding2] (this document is highly recommended  
 336 reading).

337 2. MAY support ID-WSF 1.2

338 3. An implementation MUST support the following sec mechs, see [SecMech2]:

339 - "urn:liberty:security:2005-02:TLS:Bearer"

340 - "urn:liberty:security:2006-08:TLS:SAMLV2" (Holder-of-Key, HoK)

341 A deployment MAY, as a configuration option, choose either.

342 4. MAY support following sec mechs for testing, but MUST NOT permit their use in production environ-  
 343 ments:

344 - "urn:liberty:security:2005-02:null:Bearer"

345 - "urn:liberty:security:2006-08:null:SAMLV2" (Holder-of-Key, HoK)

346 5. MAY support other TLS [RFC2246] based sec mechs, including ClientTLS.

347 6. MUST NOT permit non-TLS sec mechs in production environments

348 7. Implementations SHOULD be Liberty Alliance certified, see [IDWSF2SCR].

349 8. Implementations MUST support <ProcessingContext> "urn:liberty:sb:2003-08:ProcessingContext:Simulate"  
 350 SOAP header and implement a "dry-run" feature using it. A deployment MAY, as a configuration op-  
 351 tion, enable this feature. Partially satisfies Reqs. *D1.2-12.13-Vfy* and *D1.2-12.16-OnlineTst*.

352 9. An implementation MUST support a health check feature. We RECOMMEND that the health check  
 353 uses the "dry-run" feature mentioned in the previous item.

354 10. <sbef:Framework> SOAP header MUST be supplied and MUST have version XML attribute with  
 355 value "2.0"

356 11. <wsse:Security> SOAP header MUST be supplied

357 12. <wsu:TimeStamp> MUST be included in the <wsse:Security> SOAP header.

358 13. <a:MessageID> SOAP header MUST be included in all messages.

359 14. <a:RelatesTo> SOAP header MUST be included in all responses, unless response is an unsolicited  
 360 (spontaneous, without request) response. Including <a:RelatesTo> is especially important from audit  
 361 trail perspective so that pledges in the request can be linked to the data and obligations delivered in  
 362 the response. This rule satisfies message correlation requirement. This rule upgrades the SHOULD of  
 363 [SOAPBinding2], p.23, ll.818-822, to MUST.

- 364 15. <a:ReplyTo> SOAP header MUST be included in all requests and MUST have value <http://www.w3.org/2005/03/a>
- 365 16. <a:FaultTo> SOAP header MUST NOT be supplied. All faults are sent to <a:ReplyTo> address, i.e.  
366 in the same HTTP request-response pair.
- 367 17. <b:Sender> SOAP header MUST be included in each web service message. [[SOAPBinding2](#)] section  
368 5.9, pp.21-22, is vague about when this is needed. To simplify matters we make it always mandatory.<sup>1</sup>
- 369 18. Request-Response message exchange pattern MUST be supported.

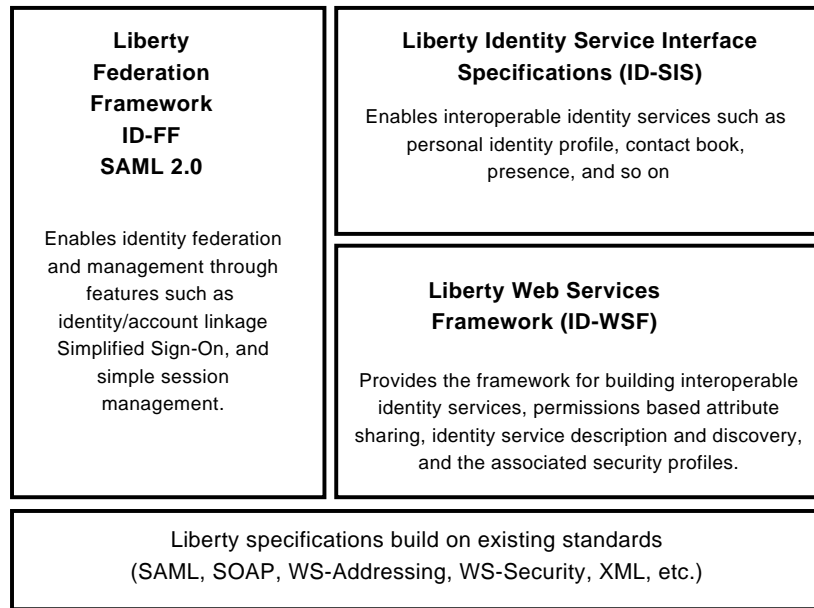


Figure 2.1: Liberty Alliance Architecture.

370

### 371 2.2.3 Bare WS-Security Header or Simplified ID-WSF

- 372 1. SHOULD NOT use, as many important security features such as message correlation, replay detection,  
373 and identification of endpoints are not supported by this mechanism.
- 374 2. Document resultant limitations if not implementing full ID-WSF.
- 375

### 376 2.2.4 WS-Trust

- 377 • MAY support [[WSTrust](#)] in general, but MUST support if deploying the particular case of accessing  
378 external Credential Validation Service, per [[ChadwickSu09](#)]

379 We have not validated whether it is possible to implement TAS<sup>3</sup> architecture using WS-Trust. Clearly  
380 WS-Trust can be used as a token exchange protocol, but for this to be interoperable heavy profiling is  
381 needed. Users and advocates of WS-Trust should undertake to write such profile.

### 383 2.2.5 RESTful Approach

384 MAY support. We RECOMMEND support on basis of OAuth [[OAUTH](#)] and OAuth WRAP [[Tom09](#)],  
385 but implementers should take in account security advisories published on [oauth.net](http://oauth.net) web site. OAuth  
386 WRAP is still immature as of this writing (Nov. 2009) and can not be recommended for production use.

387 We have not validated whether it is possible to implement TAS<sup>3</sup> architecture using RESTful approach.

<sup>1</sup>If HoK sec mech is used, the sender can generally be inferred even without this header and some implementations of ID-WSF 2.0 actually do this. However, this has caused interoperability problems, hence TAS3 tightens the rule.

388 RESTful enablement is nice to have, but should not compromise elegance of the SOAP solution and  
 389 may be less capable (i.e. it is enough that the RESTful approach solves front channel use cases). RESTful  
 390 approach may support more economical token formats such as Simple Web Token (SWT) [[Hardt09](#)].

392 TAS<sup>3</sup> project plans to address RESTful binding in future work during 2010.

## 393 2.2.6 Message Bus Approach

394 We see deploying TAS<sup>3</sup> services on message bus architecture as feasible. This will be investigated in a  
 395 future iteration of this deliverable.

## 397 2.3 Authorization Systems

398 This section addresses Reqs. *D1.2-2.19-AzCredi* and *D1.2-2.20-Az*.

399 Authorization systems are extensively covered in [[TAS3D71IdMAnAz](#)].

### 401 2.3.1 Authorization Queries

- 402 1. MUST support XACML 2.0 [[XACML2](#)] request-response contexts for authorization queries
- 403 2. MAY support other versions of XACML
- 404 3. MAY support XACML policy language
- 405 4. MUST support XACML SAML Authorization Query extension [[XACML2SAML](#)] in order to allow
- 406 policies to be dynamically passed to the PDP

407 All communication between the PEP and PDP will be using SOAP based XACML SAML profile. This  
 408 profile is mostly independent of rules language. Thus the PERMIS and trust and reputation language  
 409 specificity will be mostly contained within the PDPs themselves. The only exception is the obligation  
 410 vocabulary which must be understood by the distributed Obligations Services and therefore needs to be  
 411 standardised. This is a major effort that has already been started in the TAS<sup>3</sup> project. On the other hand,  
 412 the sticky policies, which will be passed over the wire in the protocol exchange, will be engineered such  
 413 that they transparently pass from the data store to the appropriate field of the XACML request without the  
 414 PEP proper really having to understand them.

### 416 2.3.2 Policy Languages

417 TAS<sup>3</sup> does not mandate any specific policy language. However, consider following possibilities:

- 418 1. PDP SHOULD support XACML 2.0 policy language [[XACML2](#)]
- 419 2. PDP MAY support PERMIS 5.0 policy language
- 420 3. PDP MAY support P3P policy language
- 421 4. PDP MAY support PrimeLife privacy policies
- 422 5. PEP, PDP, and Obligations Service MAY support SOL1, see section 2.11, for obligations
- 423 6. CVS MAY support PERMIS Policy CVS Schema (cf. [[TAS3D71IdMAnAz](#)] Appendix 2)

## 425 2.4 Trust and Security Vocabularies

426 Usage of ontologies in TAS<sup>3</sup> is thoroughly addressed in [[TAS3D22UPONTO](#)], which will map some of  
 427 these vocabularies.

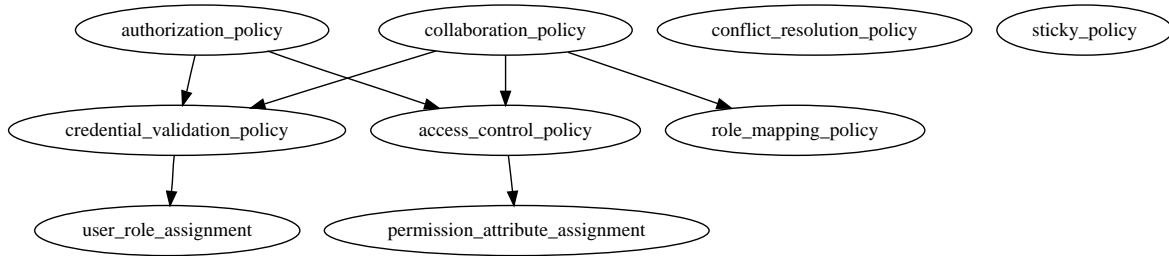


Figure 2.2: Hierarchy of policies

428

#### 429 2.4.1 Levels of Authentication (LoA)

430 TAS<sup>3</sup> recommends the use of the NIST 4 levels of assurance as described in [NIST-SP800-63] and  
 431 profiled in [SAML2LOA].

432 TAS<sup>3</sup> is working on determining whether and how to support LoA schemes of various European coun-  
 433 tries.

#### 435 2.4.2 Vocabularies for Authorization

436 Some work has been done in RADIUS [RFC2138] and Diameter [RFC3588].  
 437 [SAML2context] is mainly about authentication, but authorization is also touched.  
 438 This section will be expanded in a future version of this document.

#### 440 2.4.3 Vocabularies for Basic Attributes (PII)

441 Use of following vocabularies of PII is RECOMMENDED:

- 442 • LDAP inetOrgPerson [RFC2798]
- 443 • Liberty Personal Profile specification [IDPP]
- 444 • X.500 standards, such as [X520] and [X521]. See also [RFC2256].

445 This section will be expanded in a future version of this document.

#### 447 2.4.4 Discovery Vocabularies

448 Main vocabulary for discovery is the Service Type taxonomy described in [Disco2]. This taxonomy is  
 449 complemented by discovery options that further describe the service. This vocabulary SHOULD be used  
 450 when applicable.

451 Each Liberty service specifies its own Service Type value as well as a number discovery options. For  
 452 example, see [IDDAP], [IDPP], or [DST21].

453 This section will be expanded in a future version of this document.

#### 455 2.4.5 Security and Trust Vocabularies

456 See [SAML2context] and [SecMech2] for a vocabulary of security mechanisms that MUST be used  
 457 when applicable.

458 This section will be expanded in a future version of this document.

#### 460 2.4.6 Audit Vocabularies

461 Audit events from RADIUS [RFC2139] and Diameter [RFC3588] are RECOMMENDED for use where  
 462 applicable.

463 This section will be expanded in a future version of this document. As audit is active research topic, we  
 464 benefit from the research during the TAS<sup>3</sup> project to specify this section in detail in the final version of  
 465 this document.

## 467 2.5 Realization of the Discovery Function

- 468 • MUST support Liberty ID-WSF 2.0 Discovery Service specification [Disco2]
- 469 • MAY support [Disco12]
- 470 • MAY support UDDI, however this may require significant extensions to UDDI. Such extensions  
 471 would need to be profiled.

472 See [NexofRA09], section 5.4 "The Overview-Model", fig 18, for a view of the interaction between  
 473 service registration and service discovery. Unfortunately the referred document fails to recognize the need  
 474 for per-identity service registrations, unless the oblique reference, where no difference is made between  
 475 service requester entity and the data subject, in section 5.4.4 "Service Discovery", counts.

## 477 2.6 Realization of the Credentials and Privacy Negotiator Function

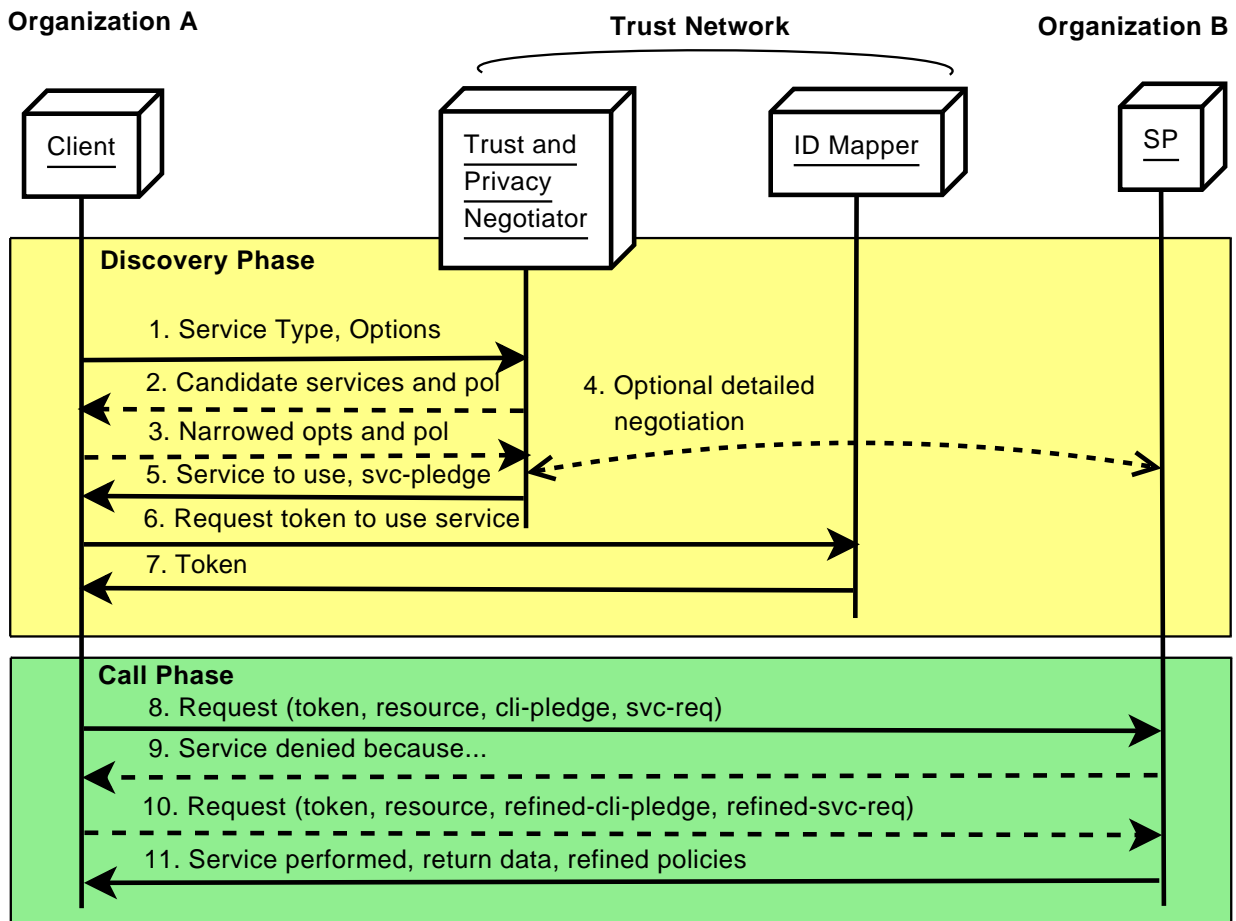


Figure 2.3: Credentials and Privacy Negotiation and Discovery steps

478 Credentials and Policy Negotiation generally takes authentication and identification of all parties for  
 479 granted, but then computes a trust score which typically governs the access control decisions.

### 481 2.6.1 Discovery in Credentials and Privacy Negotiation

482 In this model both "Credentials and Privacy Negotiator" and "ID Mapper" are implemented as parts of  
 483 Discovery Service.

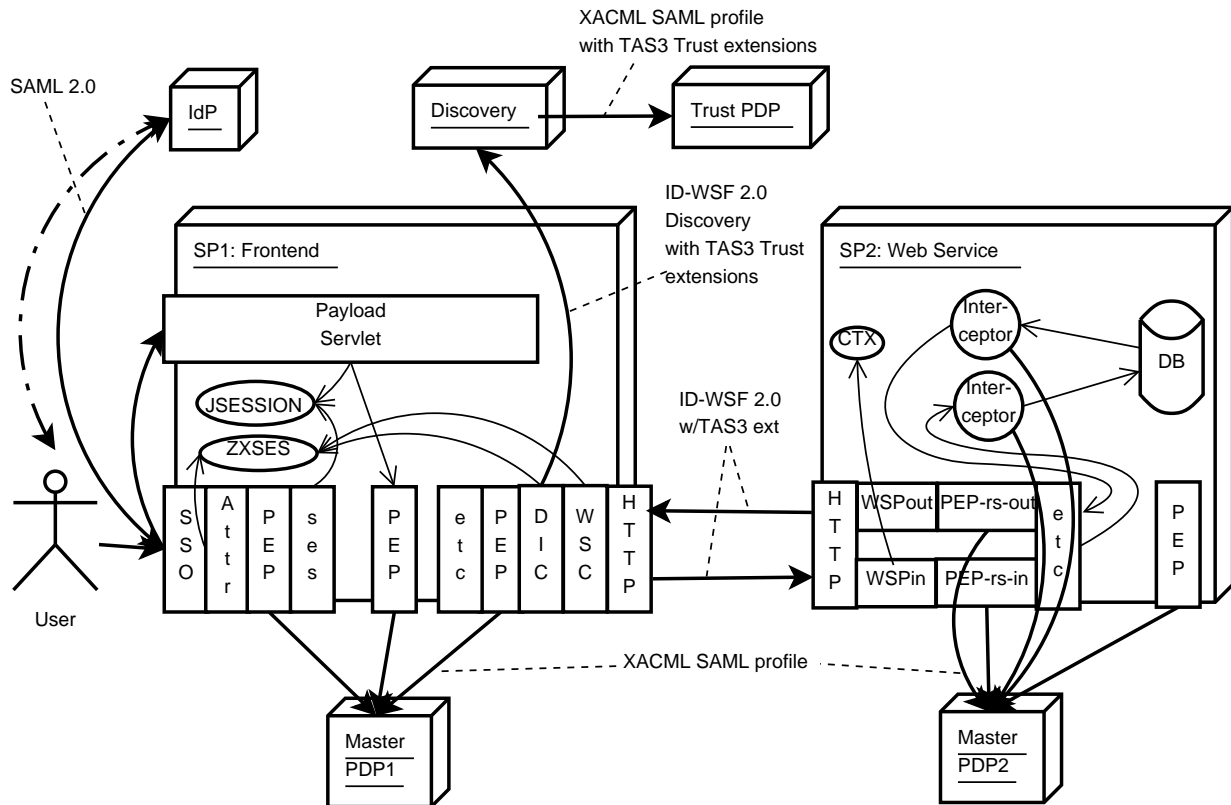


Figure 2.4: A deployment architecture for Credentials and Privacy Negotiation and Discovery

484

## 2.6.2 Frontend Credentials and Privacy Negotiation

485

In future work we will address user giving input to Credentials and Privacy Negotiation.

486

## 2.6.3 Components of Credentials and Privacy Negotiator

488

1. Service Requestor (SR) discovers the location of the User's Credentials and Privacy Negotiator Agent (U-CPNA) and a candidate list of Web Service Providers (WSPs).

489

2. SR passes the candidate list to the U-CPNA.

491

3. U-CPNA discovers the location of user's attribute aggregator.

492

4. U-CPNA obtains a token with user's pseudonym at the Attribute Aggregator.

493

5. U-CPNA obtains necessary credentials for the user from the Attribute Aggregator. Attribute Aggregator, in turn may contact Attribute Authorities to obtain the credentials. Each such contact involves its own web service call, with discovery, IDMap, and actual web service calls, each with appropriate authorization steps. This complexity is not shown in the diagram.

494

495

496

497

6. U-CPNA engages in credentials and privacy negotiation with the WSP's Credentials and Privacy Negotiation service.

498

499

7. Once U-CPNA returns the chosen WSP, the SR obtains a token for calling the WSP.

500

8. Finally the actual web service call is realized (with appropriate authorization steps, not shown in the diagram).

501

502

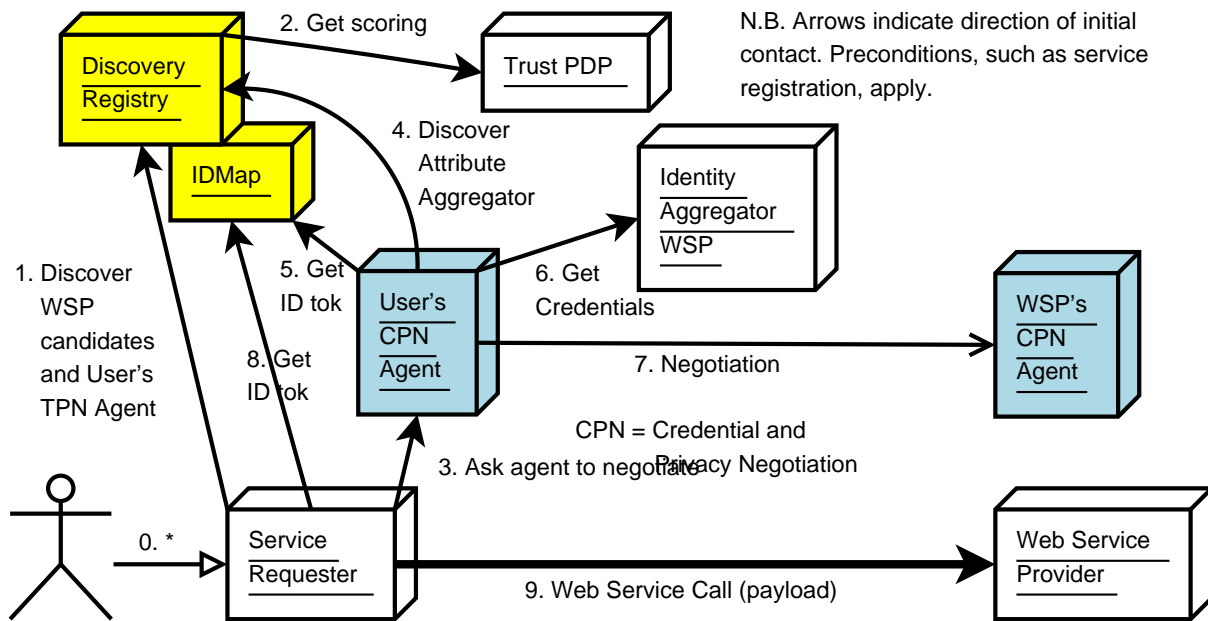


Figure 2.5: Credentials and Privacy Negotiation Components

503 Some variants and optimizations to this basic flow are possible. One obvious variant is to merge the  
 504 calls to Discovery Registry and IDMapper. Liberty Alliance Discovery Service [Disco2] effectively uses  
 505 this optimization.

506 Another, perhaps more significant, optimization is to integrate the credentials and privacy negotiation  
 507 under the Discovery Service. In this scenario, the U-CPNA is called from the midst of the discovery  
 508 process. This reduces steps and may allow the discovery process to use criteria from the credentials and  
 509 privacy negotiation.

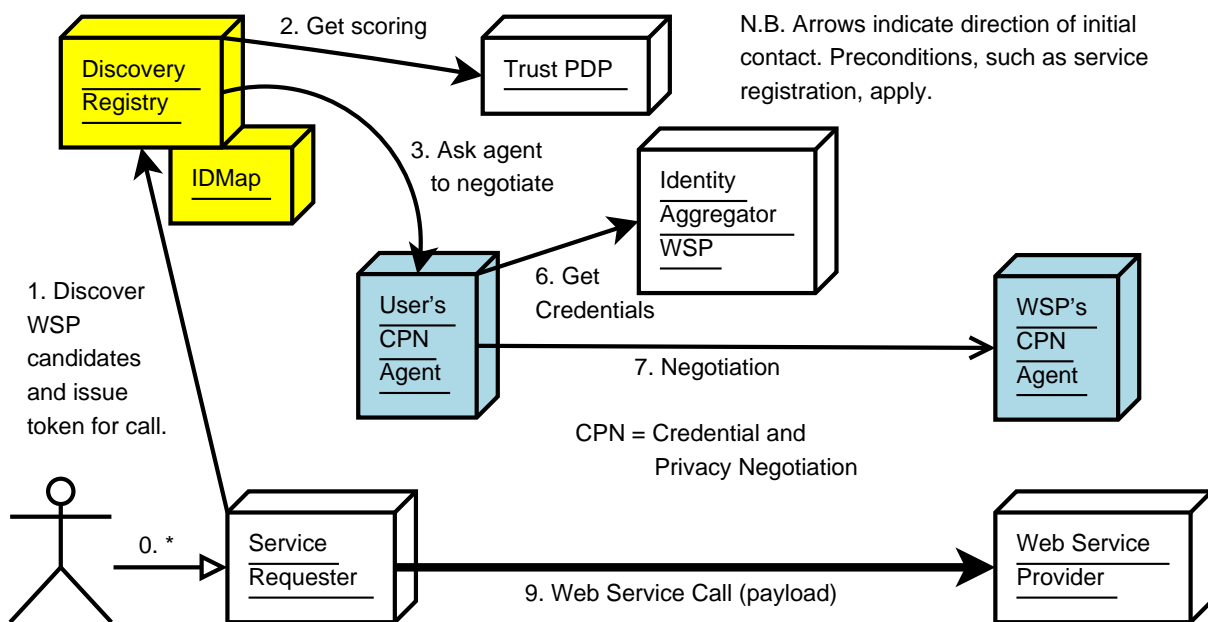


Figure 2.6: Credentials and Privacy Negotiation optimized flow

510 **1** Service Requestor (SR) discovers Web Service Provider (WSP).

511 **2** Discovery passes the candidate list to the U-CPNA. Discovery can also pass the End Point Reference  
 512 (EPR), which includes a token with pseudonym for the call, to the Attribute Aggregator.

513 **5** U-CPNA obtains necessary credentials for the user from the Attribute Aggregator in same way as in  
 514 unoptimized case.

515 **6** U-CPNA engages in credentials and privacy negotiation with the WSP's Credentials and Privacy Nego-  
 516 tiation service.

517 **8** The discovery service returns to SR the EPR of the WSP. Finally the actual web service call is realized.  
 518

## 519 **2.6.4 Protocol between Service Requester and the Credentials and Privacy** 520 **Negotiation Agent**

521 Service Requester invokes the User's Credentials and Privacy Negotiation Agent as a regular web ser-  
 522 vice. The body of the call needs to express

- 523 • Candidate list

## 525 **2.6.5 Protocol between Credentials and Privacy Negotiation Agent and At-** 526 **tribute Aggregator**

527 User's Credentials and Privacy Negotiation Agent invokes user's Attribute Aggregator as a regular web  
 528 service. The body of the call needs to express what credentials are desired and the body of the response  
 529 must be able to pass multiple credentials.

## 531 **2.6.6 Protocol between Credentials and Privacy Negotiation Agent and Ser-** 532 **vice**

533 The protocol to realise the credentials and privacy negotiation functionality has yet to be finalised.  
 534 Candidate protocols are:

- 535 i. the one used by TrustBuilder 2 [[TrustBuilder2](#)]
- 536 ii. one based on the Web Service Profile of XACML [[Anderson07](#)] as enhanced by [[Mbanaso09](#)]
- 537 iii. one based on an enhanced Liberty Discovery Service [[Disco2](#)]

538 Whichever protocol is finally chosen it must be able to support a ceremony to gaining incremental levels  
 539 of mutual trust. The Web GUI of the Front End MUST support the ceremony.

## 541 **2.7 Using Trust Scoring in Discovery**

542 The Trust Scoring is available from the Trust PDP component. As PDPs use XACML protocol, which  
 543 natively does not have ability to convey anything else than Permit or Deny decision and associated obli-  
 544 gations, we profile the second level XACML <StatusCode> to carry the ranking information: the Value  
 545 XML attribute holds a URN prefix, identifying the trust ranking scheme, followed by actual raning in the  
 546 syntax specified by the scheme.

### 547 **Example**

```
548 <StatusCode Value="urn:oasis:names:tc:xacml:1.0:status:ok">
549   <StatusCode Value="urn:eu.tas3.trustranking:eu.tas3.trustpdp.centralityrtm.RTMEngine:CENTRALITY">
550 </StatusCode>
```

551

## 552 2.8 Realization of the Audit and Dashboard Function

553

### 554 2.8.1 Audit Event Bus

555 Satisfies Req. *D1.2-9.5-Trail*.

556 Tentative protocol choice (in order of preference):

557 1. AMQP [[AMQP06](#)]

558 2. Liberty Accounting Service [[AcctSvc](#)] with subscriptions and notifications [[SUBS2](#)] and [[DesignPat](#)].

559 3. Diameter [[RFC3588](#)]

560 4. RADIUS [[RFC2138](#)]

561 5. Apache Muse

562 Whichever transport is chosen, the actual audit records are packaged as OpenXDAS messages (see:  
563 [openxdas.sourceforge.net](#)).

### 565 2.8.2 Audit Event Ontology

566 • Enumeration of mandatory edit events according to some standard

567 - RADIUS and Diameter communities have defined at least some messages

568 • ZXID logging documentation [[ZXIDREADME](#)] provides an idea, at least applicable to SSO

569

### 570 2.8.3 Dashboard Function

571 • Dashboard should also realize the "PII Consent Service" or "Privacy Manager" at large.

572 • SHOULD support Liberty Interaction service [[Interact2](#)]

573

### 574 2.8.4 User Interaction

575 User interaction is needed for consent questions and possibly even soliciting additional data during back  
576 channel web service calls. Interaction can be realized using two different mechanisms

577 a. Liberty Interaction service [[Interact2](#)] where a web services call is made to the interaction service. This  
578 service is often colocated with the Dashboard.

579 b. The web service returns special SOAP fault requesting redirection to interaction URL.

580 Special attribute for interaction iFrame URL.

## 582 2.9 Realization of Delegation Function

583 The Delegation Service functionality is described in section 6 of D7.1. The protocols that this will use  
584 will be described in the next version of the current deliverable.

585

## 586 2.10 Attribute Authorities

587 TAS<sup>3</sup> network may contain various attribute authorities. Every Identity Provider may act as an attribute  
 588 authority by including <AttributeStatement>, see [SAML2core], in the single sign-on assertions that  
 589 it emits. This constitutes an attribute push mechanism.

590 Problem with a push mechanism is knowing which attributes to push. A possible solution is for the Front  
 591 End to express its attribute needs using a SAML extension, such as [Kellomaki08]. However, usually a  
 592 better solution is to implement pull model Attribute Authority, i.e. the attribute authority is simply a web  
 593 service.

594 There are several ways of implementing a data web service. [SAML2prof] specifies AttributeQuery  
 595 protocol, but does not adequately specify the transport binding and peer authentication. TAS<sup>3</sup> attribute au-  
 596 thority SHOULD support [SAML2prof] AttributeQuery protocol using TAS<sup>3</sup> SOAP binding, see section  
 597 2.2.2.

598 Other data web services, such as ID-DAP [IDDAP] over TAS<sup>3</sup> SOAP binding, MAY be supported. A  
 599 deployment may also make local or proprietary arrangements for accessing a non TAS<sup>3</sup> attribute authority,  
 600 e.g. using LDAP [RFC2251] or WebDAV with file containing attribute certificate or SAML attribute  
 601 assertion.

## 603 2.11 TAS<sup>3</sup> Simple Obligations Language (SOL)

604 TAS<sup>3</sup> Architecture foresees that a Service Requester needs to express obligations and policies that it is  
 605 willing and able to respect, and on the other hand the personal data will have associated with it obligations  
 606 and policies ("sticky policies") under which the data can be or is released.

607 In general the obligations and sticky policies can be expressed in any convenient language. Unfortu-  
 608 nately no standard language has emerged in the industry for this type of application despite many being  
 609 proposed. TAS<sup>3</sup> is committed to supporting multiple such languages, but for purposes of pilots and other  
 610 simple applications we define "TAS<sup>3</sup> Simple Obligations Language n<sup>o</sup>1" (SOL1) with potential future  
 611 versions to follow.

612 SOL obligations MAY be used in XACML obligations as described in [TAS3D71IdMAnAz]. In partic-  
 613 ular, D7.1 Appendix A1.2 provides an example. In short, they MUST appear in an Obligation/AttributeAssignment  
 614 element. When passed in <b:UsageDirective>, <xa:Obligation> element MUST be used as a wrap-  
 615 per. Use of <xa:Obligation> element as a wrapper in other XML contexts is RECOMMENDED.

616 N.B. Since SOAP headers in TAS<sup>3</sup> are generally signed, the <b:UsageDirective> header  
 617 constitutes signed pledge to honour the obligations. This is similar to Signed Acceptance of  
 618 Obligations (SAO) concept of Obligation of Trust (OoT) protocol described in [Mbanaso09]  
 619 et al. Put another way, the pledge expresses the Capabilities. We effectively optimize the OoT  
 620 Protocol Scheme (sec 3.2) by avoiding iterative discovery of capabilities and moving directly  
 621 to the signed pledge phase (5 in fig. 5).

622 The ObligationId XML attribute of <xa:Obligation> element is used to specify the obligations  
 623 processor (module that the PDP should invoke to evaluate the obligation). Some processors may be simple  
 624 in which case the ObligationId completely identifies the nature of the obligation.

625 When using SOL, however, the semantics of the obligation depend on the actual SOL expressions passed  
 626 in the <xa:AttributeAssignment> child element of <xa:Obligation>. In this case the ObligationId  
 627 merely identifies the obligations processing engine. The SOL1 obligations processor is identified by  
 628 ObligationId value "urn:tas3:sol1". The actual SOL1 expressions are held in <xa:AttributeAssignment>  
 629 elements with following AttributeId XML-attributes:

630 **urn:tas3:sol1:pledge** Obligations that WSC pledges to honour if it receives them in any response  
 631 data.

632 **urn:tas3:sol1:require** Obligations that the emitting party requires to be honoured. Typically  
 633 this is used to attach obligations to the data that is returned.

634 There **MUST** only be one `<xa:AttributeAssignment>` with each `AttributeId`, i.e. there can only  
 635 be zero, one, or two `<xa:AttributeAssignment>` elements in `<xa:Obligation>` element. There **MUST**  
 636 only be one `<xa:Obligation>` element with `ObligationId` "urn:tas3:sol1" and there **MUST** only be one  
 637 `<b:UsageDirective>` in the SOAP message.

638 The `DataType` XML attribute of the `<xa:AttributeAssignment>` **MUST** always have value  
 639 "http://www.w3.org/2001/XMLSchema#string". The `FulfillOn` XML attribute of `<xa:Obligation>`  
 640 element **SHOULD**, in absence of more specific guidance, be set to "Permit".

641 The `urn:tas3:sol:vers` Query String parameter allows for versioning of the obligations language.  
 642 The actual obligations are expressed using URL Query String Syntax with attribute value pairs expressing  
 643 the obligations. Newline (0x0a) **MAY** be used as separator instead of an ampersand. Should escaping be  
 644 needed, the URL encoding **MAY** be used.

#### 645 **Example**

```
646 <b:UsageDirective id="USE">
647   <xa:Obligation ObligationId="urn:tas3:sol1" FulfillOn="Permit">
648     <xa:AttributeAssignment
649       AttributeId="urn:tas3:sol1:pledge"
650       DataType="http://www.w3.org/2001/XMLSchema#string">
651       urn:tas3:sol:vers=1
652       urn:tas3:sol1:delon=1255555377
653       urn:tas3:sol1:use=urn:tas3:sol1:use:purpose
654       urn:tas3:sol1:share=urn:tas3:sol1:share:group
655       urn:tas3:sol1:repouse=urn:tas3:sol1:repouse:oper
656     </xa:AttributeAssignment>
657   </xa:Obligation>
658 </b:UsageDirective>
```

659 As can be seen from the example, the attributes are actually URNs and each attribute tends to express  
 660 an obligation that is required by data or that the Requester promises to honour.

### 662 **2.11.1 SOL1 Query String Attributes**

663 **urn:tas3:sol:vers** Identifies the version of SOL. Always "1" for SOL1.

664 **urn:tas3:sol1** Special value reserved to be used as `ObligationId` or in general to identify this  
 665 dialect of SOL.

666 **urn:tas3:sol1:pledge** Special value reserved to be used as `AttributeId`

667 **urn:tas3:sol1:require** Special value reserved to be used as `AttributeId`

668 **urn:tas3:sol1:use** How information can or will be used and shared. A comma separated list of  
 669 enumerators in the order of principally intended use (ordered here, in our opinion, from least ag-  
 670 gressive to more aggressive as indicated; however this ordering is subjective and other opinions may  
 671 exist). The `urn:tas3:sol1:use:purpose` should be favoured over `urn:tas3:sol1:use`, unless  
 672 the vague meaning of `urn:tas3:sol1:use` is desired.

673 **urn:tas3:sol1:use:transaction (0)** Information will only be used for the transaction  
 674 for which it was collected

675 **urn:tas3:sol1:use:session (1)** Information will only be used within the current session

676 **urn:tas3:sol1:use:user (2)** Information can be used in the user's other sessions in the  
 677 same app

678 **urn:tas3:sol1:use:forpurpose (3)** Information will be used only for the purpose it was  
 679 collected, in abstract. This usage is discouraged. Instead the specific purpose should be speci-  
 680 fied using format

681 urn:tas3:sol1:use:purpose=business-process-model-id; or  
 682 urn:tas3:sol1:use:purpose=business-process-instance-id

683 These two forms allow the obligation to be tied into the model in abstract, or to the specific  
 684 business process instance in particular, e.g. for exceptional processing such as Break-the-  
 685 Glass.

686 **urn:tas3:sol1:use:serveranon (4)** Information can be used by other processes on same  
 687 server as long as user is not explicitly identified

688 **urn:tas3:sol1:use:serverident (5)** Information can be used by other processes on same  
 689 server (user may be identified)

690 **urn:tas3:sol1:use:appanon (6)** Information can be used by the application towards other  
 691 purposes as long as the user is not explicitly identified

692 **urn:tas3:sol1:use:appid (7)** Information can be used by the application towards other  
 693 purposes (user may be identified)

694 **urn:tas3:sol1:use:organon (8)** Information can be used by the organization for other  
 695 nonmarketing purposes as long as the user is not explicitly identified

696 **urn:tas3:sol1:use:orgident (9)** Information can be used by the organization for other  
 697 nonmarketing purposes (user may be identified)

698 **urn:tas3:sol1:use:mktanon (10)** Information can be used by the organization for market-  
 699 ing purposes as long as the user is not explicitly identified

700 **urn:tas3:sol1:use:mktident (11)** Information can be used by the organization for mar-  
 701 keting purposes (user may be identified)

702 **urn:tas3:sol1:use:grpanon (12)** Information can be used within the business group for  
 703 other nonmarketing purposes as long as the user is not explicitly identified

704 **urn:tas3:sol1:use:grpident (13)** Information can be used within the business group for  
 705 other nonmarketing purposes (user may be identified)

706 **urn:tas3:sol1:use:grpmtanon (14)** Information can be used within the business group  
 707 for marketing purposes as long as user is not explicitly identified

708 **urn:tas3:sol1:use:grpmtident (15)** Information can be used within the business group  
 709 for marketing purposes (user may be identified)

710 **urn:tas3:sol1:use:shareanon (16)** Information can be shared with anyone for other non-  
 711 marketing purposes as long as the user is not explicitly identified

712 **urn:tas3:sol1:use:shareident (17)** Information can be shared with anyone for other  
 713 nonmarketing purposes (user may be identified)

714 **urn:tas3:sol1:use:sharemtanon (18)** Information can be shared with anyone for mar-  
 715 keting purposes as long as user is not explicitly identified

716 **urn:tas3:sol1:use:sharemtident (19)** Information can be shared with anyone for mar-  
 717 keting purposes (user may be identified)

718 **urn:tas3:sol1:use:anyall (20)** Information can be used for any and all purposes without  
 719 restriction.

720 **urn:tas3:sol1:use:purpose** Specific business process that is allowed to use the data. This can  
 721 be specified either as abstract business-process-model-id or as business-process-instance-id. For  
 722 example:

723 urn:tas3:sol1:use:purpose=business-process-model-id; or  
 724 urn:tas3:sol1:use:purpose=business-process-instance-id

725 These two forms allow the obligation to be tied into the model in abstract, or to the specific business  
 726 process instance in particular, e.g. for exceptional processing such as Break-the-Glass.

727 **urn:tas3:sol1:delon** Delete data on as Unix seconds since epoch. This obligation effectively  
 728 allows control of data retention, but instead of being expressed in relative terms, it is expressed in  
 729 absolute terms that are legally easier to interpret.

730 **urn:tas3:sol1:retention** Maximum data retention period as Unix seconds. This obligation is  
 731 meant for database storage. Upon act of data access, retention should be converted to delon using  
 732 current wall clock time.

733 **urn:tas3:sol1:certdel** Certify deletion by legally binding report to the audit bus.

734 **urn:tas3:sol1:preauth** Before each use of the data, user's explicit consent - preauthorization -  
 735 has to be obtained. Value specifies where to obtain preauthorization.

736 **urn:tas3:sol1:callback** When about to use data, call back to the user for opportunity to modify  
 737 the data, or deny it. Value specifies where to call back.

738 **urn:tas3:sol1:repouse** Report use to the audit bus. Comma separated list of enumerators:

739 **urn:tas3:sol1:repouse:never** No need to report use (seldom appears)

740 **urn:tas3:sol1:repouse:all** Report any and all use

741 **urn:tas3:sol1:repouse:oper** Report operational use, but not statistical or administrative  
 742 use.

743 **urn:tas3:sol1:repouse:stat:immed** Report use in near real time. for day need to be  
 744 reported, if there was any use.

745 **urn:tas3:sol1:repouse:stat:daily** No need to report individual use, but summary  
 746 statistics for day need to be reported, if there was any use.

747 **urn:tas3:sol1:repouse:stat:weekly** No need to report individual use, but summary  
 748 statistics for week need to be reported, if there was any use.

749 **urn:tas3:sol1:repouse:stat:monthly** No need to report individual use, but summary  
 750 statistics for month need to be reported, if there was any use.

751 **urn:tas3:sol1:repouse:stat:quarterly** No need to report individual use, but sum-  
 752 mary statistics for quarter (last 3 months) need to be reported, if there was any use.

753 **urn:tas3:sol1:repouse:stat:semestral** No need to report individual use, but sum-  
 754 mary statistics for semester (last 6 months) need to be reported, if there was any use.

755 **urn:tas3:sol1:repouse:stat:yearly** No need to report individual use, but summary  
 756 statistics for year need to be reported, if there was any use.

757 If no **urn:tas3:sol1:repouse:stat** is specified, default is **urn:tas3:sol1:repouse:stat:immed**.

758 If conflicting enumerators are specified, the most strict one applies.

759 **urn:tas3:sol1:xborder** Enumerator describing what sort of cross border data sharing can occur:

760 **urn:tas3:sol1:xdom:eu** Only within EU common market.

761 **urn:tas3:sol1:xdom:safeharbour** Common market and safe harbour participants

762 **urn:tas3:sol1:license** Use of information is subject to license specified in the value part. The  
 763 value part should be either URL to online accessible license text, or it should be a URN pointing to  
 764 a well known license.

765 The general assumption is that the license terms are either well known to the system (and pro-  
 766 grammed in) or machine readable. While the user may have to consent to the license at some level,  
 767 it is not meant that this license reference be displayed to user and he required to read and consent  
 768 on the spot.

769 **urn:tas3:sol1:contract-fwk** Framework or governance contract identifier.

770 **urn:tas3:sol1:contract** Contract identifier.

771 **urn:tas3:sol1:contract-sub** Subcontract or amendment identifier

772 **urn:tas3:sol1:contract-part** Part, exhibit, annex, or clause identifier.  
 773

## 774 2.11.2 Matching Pledges to Sticky Policies and Obligations

775 When delivering response to data request, the Responder outbound PEP compares the pledges that were  
 776 received in the request and checks that the sticky policies and obligations that are attached to the data  
 777 coming from the backend repository can be satisfied given the pledges. This ensures that the Responder  
 778 will never ship out data unless the Requester has clearly committed itself to respect the sticky policies and  
 779 obligations.

### 780 **Example**

781 Consider the following request

```
782 <e:Envelope>
783   <e:Header>
784     <!-- WS-Addressing headers and wsse:Security with DSIG not shown -->
785     <b:UsageDirective id="USE">
786       <xa:Obligation ObligationId="urn:tas3:sol1" FulfillOn="Permit">
787         <xa:AttributeAssignment
788           AttributeId="urn:tas3:sol1:pledge"
789           DataType="http://www.w3.org/2001/XMLSchema#string">
790           urn:tas3:sol:vers=1
791           urn:tas3:sol1:delon=1255555377
792           urn:tas3:sol1:use=urn:tas3:sol1:use:purpose
793           urn:tas3:sol1:share=urn:tas3:sol1:share:group
794           urn:tas3:sol1:repouse=urn:tas3:sol1:repouse:oper
795         </>
796       </>
797     </>
798   </>
799   <e:Body id="BDY">
800     <idhrxml:Query>...</></></>
```

801 Now, backend returns the following data

```
802 <dataItem id="1">
803   <tas3sol:Obligations xmlns:tas3sol="http://tas3.eu/tas3sol/200911/">
804     urn:tas3:sol:vers=1
805     urn:tas3:sol:delon=1255555378
806     urn:tas3:sol1:use=urn:tas3:sol1:use:transaction
807   </>
```

```

808     <data>value</>
809 </>
810
811 <dataItem id="2">
812   <tas3sol:Obligations xmlns:tas3sol="http://tas3.eu/tas3sol/200911/">
813     urn:tas3:sol:vers=1
814     urn:tas3:sol:delon=1255555376
815     urn:tas3:sol1:use=urn:tas3:sol1:use:purpose
816     urn:tas3:sol1:repose=urn:tas3:sol1:repose:all
817   </>
818   <data>value</>
819 </>
820
821 <dataItem id="3">
822   <tas3sol:Obligations xmlns:tas3sol="http://tas3.eu/tas3sol/200911/">
823     urn:tas3:sol:vers=1
824     urn:tas3:sol:delon=1255555378
825     urn:tas3:sol1:use=urn:tas3:sol1:use:purpose
826     urn:tas3:sol1:repose=urn:tas3:sol1:repose:oper, repose=urn:tas3:sol1:repose:stat:weekl
827   </>
828   <data>value</>
829 </>
    
```

830 The first data item would have to be filtered out because its usage policy is "transaction" while requester  
 831 pledged usage for intended "purpose". Intended purpose can span many transactions, therefore its broader  
 832 that the allowed use. Note that the `delon` constraint would be compatible with the request.

833 The second data item has to be filtered out for two reasons: (i) its `delon` is stricter than what requester  
 834 pledged, and (ii) the `repose` constraint is more onerous than requester is willing to perform.

835 The third data item's obligations are compatible with the requester's pledges. It is returned to the  
 836 requester.

837 N.B. This is just an example. The way in which the obligations are attached to the data can be  
 838 quite different from the illustrated, e.g. internal C data structure rather than XML. It is also  
 839 possible that obligations are not stored with the data, but rather generated by a PDP based on  
 840 data dependent sticky-policies.

841 Once the Responder Outbound PEP has filtered the data, it is sent, with the obligations, to Requester  
 842 which MAY pass the obligations to Obligations Service for enforcement.

### 844 2.11.3 Passing Simple Obligations Dictionaries Around

845 While in SOL1 the set of enumerators is fixed and with fixed meaning which is hardwired to the simplest  
 846 PEP implementations, we foresee users inventing additional attributes and enumerators. This raises the  
 847 need for the PEP implementations to be configurable or somehow understand the new enumerators on  
 848 basis of their semantics.

849 Such configurations and online semantics passing can be achieved with Simple Obligations Dictionaries  
 850 (SODs), which effectively allow the semantics to be declared. The dictionary can be stored in a configura-  
 851 tion file, and we provide SOL1 standard dictionary as `sol1.sod` (which you should not modify) and you  
 852 may be able to provide additional dictionary fragments in user editable configuration files. Alternatively,  
 853 the nonstandard dictionary fragments can be passed inline in the protocol by means of `<tas3sol:Dict>`  
 854 element.

#### 855 Example

```
856 <e:Envelope>
```

```

857 <e:Header>
858 <!-- WS-Addressing headers and wsse:Security with DSIG not shown -->
859 <b:UsageDirective id="USE">
860 <xa:Obligation ObligationId="urn:tas3:sol1" FulfillOn="Permit">
861 <xa:AttributeAssignment
862 <AttributeId="urn:tas3:sol1:pledge"
863 <DataType="http://www.w3.org/2001/XMLSchema#string">
864 urn:tas3:sol:vers=1
865 urn:tas3:sol1:delon=1255555377
866 urn:tas3:sol1:use=urn:tas3:sol1:use:purpose
867 urn:tas3:sol1:share=urn:tas3:sol1:share:group
868 urn:tas3:sol1:repose=urn:tas3:sol1:repose:oper
869 </>
870 </>
871 <tas3sol:Dict xmlns:tas3sol="http://tas3.eu/tas3sol/200911/">
872 Entities:
873 Data Subject (Agent the Data describes)
874 Data Processor (Agent that processes the Data)
875 Data (Information which is a resource under protection)
876 Organisation (a Data Processor)
877 Marketing (an Action)
878 Process (an Action of manipulating Data)
879
880 Relations:
881 Identify
882 Retain
883
884 Property
885 May (property of an action)
886 Must (property of an action)
887
888 urn:tas3:sol1:use:mktident is an enumerator of urn:tas3:sol1:use
889
890 urn:tas3:sol1:use:mktident means
891 Organization (who) - Process (action) - Data (what) - Marketing (why)
892 Organization (who) - Identify (action) - Data Subject (What)
893 </>
894 </>
895 </>
896 <e:Body id="BDY">
897 <idhrxml:Query>...</></></>

```

898 This example uses `<tas3sol:Dict>` element to define a new enumerator for `urn:tas3:sol1:use`  
899 by spelling out its semantic meaning in terms of the dictionary items (example is somewhat unrealistic  
900 because you should not repeat or redefine dictionary entries from the standard `sol1.sod`). In particular the  
901 `mktident` really is a combination of two consequences: you will receive spam and you will be identified.  
902 Thus the "means" declaration has two lines.

## 904 2.12 Realization of Sticky Policies

905 As discussed in [TAS3ARCH] section 4.1 "Protocol Support for Conveyance of Sticky Policies", En-  
906 capsulating Security Layer (ESL) is one approach for implementing sticky policies. While total encapsu-  
907 lation is possible, for already established applications protocols something lighter weight is desired. Most  
908 properties of ESL can also be implemented by a special SOAP header that references all the elements that

would have been referenced by the ESL approach. The subtle, but salient, difference is that instead of the intrusive encapsulation layer, all the relevant policy data is carried in the `<tas3:ESLPolicy>` header.

The reference is either by XML `id` attribute (preferred) or a simplified absolute XPath [[XPath99](#)].

### Example

```

913 <e:Envelope>
914   <e:Header>
915     <wsse:Security>...</>
916     <tas3:ESLPolicies mustUnderstand="1">
917       <tas3:ESLApply>
918         <tas3:ESLRef ref="#data1"/>
919         <tas3:ESLRef xpath="container/subcontainer"/>
920         <xa:Obligation ObligationId="urn:tas3:sol1">
921           <xa:AttributeAssignment
922             AttributeId="urn:tas3:sol1:require"
923             DataType="http://www.w3.org/2001/XMLSchema#string">
924             urn:tas3:sol:vers=1
925             urn:tas3:sol1:delon=1255555377
926           </xa:AttributeAssignment>
927         </xa:Obligation>
928       </tas3:ESLApply>
929       <tas3:ESLApply>
930         <tas3:ESLRef ref="#data2"/>
931         <xa:Obligation ObligationId="urn:tas3:sol1">
932           <xa:AttributeAssignment
933             AttributeId="urn:tas3:sol1:require"
934             DataType="http://www.w3.org/2001/XMLSchema#string">
935             urn:tas3:sol:vers=1
936             urn:tas3:sol1:delon=1255566666
937           </xa:AttributeAssignment>
938         </xa:Obligation>
939       </tas3:ESLApply>
940     </tas3:ESLPolicies>
941   </e:Header>
942   <e:Body>
943     <data id="data1" value="foo">
944     <data id="data2" value="bar">
945     <container>
946       <subdata value="goo"/>
947     </container>
948   </e:Body>
949 </e:Envelope>
    
```

In the above example both `id` based references to `<data>` and XPath based reference for the `<subdata>` are illustrated. It also illustrates how to apply different sticky policies (n.b. Obligation is a particularly common type of sticky policy) to different data.

## 2.13 Passing Additional Credentials in Web Service Call

The usual way to pass credentials is using an attribute assertion inside `<wsse:Security>` header. Such attribute assertion identifies the calling user. Sometimes additional credentials identifying the actual resource are passed in `<TargetIdentity>` SOAP header. However, both of these methods basically admit single credential (which can contain other credentials as attributes) typically not signed by the Requester. If Requester needs to add additional credentials, it can use `<tas3:Credentials>` element.

```

960 <e:Envelope>
961   <e:Header>
962     <wsse:Security>...</>
963     <tas3:Credentials xmlns:tas3="http://tas3.eu/tas3/200911/">
964       ... reuse XACML or SAML attribute schema
965     </tas3:Credentials>
966   </e:Header>
967   <e:Body>...</>
968 </e:Envelope>
969

```

## 970 2.14 Uniform Application Status and Error Reporting

971 Traditionally Web Service application protocols have defined their own error and status reporting mech-  
 972 anisms. TAS<sup>3</sup> standardizes the status reporting by adding a standardized SOAP header that the application  
 973 SHOULD insert if it wishes to enable some automatic TAS<sup>3</sup> processing. This is especially important for  
 974 automation of Online Compliance Testing.

975 Some ways the errors can be reported

- 976 1. Network or socket layer, e.g. drop the connection in case of a security violation. This is very extreme  
 977 response and SHOULD NOT be used normally, unless there is a genuine threat, such as suspected  
 978 Denial-of-Service (DoS) attack.
- 979 2. HTTP layer error codes. In normal operation, 200 should be used. In particular 4xx and 5xx codes  
 980 SHOULD NOT be used to indicate authorization errors deep in the application or application errors.  
 981 The HTTP error codes SHOULD generally be used for errors that are detected at web server level.
- 982 3. Application platform errors, such as stack backtraces, SHOULD NOT happen. All errors SHOULD  
 983 be trapped and appropriately reported by the application. Despite this rule, the reality of application  
 984 development means that stack traces will be output by buggy or immature software.
- 985 4. SOAP faults. Generally SOAP faults should only be used to indicate SOAP transport level errors, as  
 986 defined by SOAP and ID-WSF specifications.  
 987 The API, such as *tas3\_get\_fault()*, for creating and inspecting TAS<sup>3</sup> related SOAP faults is described  
 988 in section 3.1.13 "SOAP Fault and Status Generation and Inspection".
- 989 5. ID-WSF special headers. Some ID-WSF level errors cause an ID-WSF specific SOAP headers to be  
 990 emitted in the response.
- 991 6. TAS<sup>3</sup> error header SHOULD be used to report all TAS<sup>3</sup> and application level errors.
- 992 7. Application level error mechanisms MAY be used to report application level errors. It is RECOM-  
 993 MENDED that the application level protocols be designed to use the TAS<sup>3</sup> error headers or at least the  
 994 Liberty Utility schema defined <Status> element [DesignPat].

### 996 2.14.1 TAS<sup>3</sup> Status Header

997 The TAS<sup>3</sup> Status Header is based on the <Status> element defined in Liberty Utility Schema, see  
 998 [DesignPat].

```

999 <e:Envelope>
1000   <e:Header>
1001     <tas3:Status
1002       xmlns:tas3="http://tas3.eu/tas3/200911/"
1003       ctlpt="urn:tas3:ctlpt:app"

```

```

1004         code="OK"/>
1005     </e:Header>
1006     <e:Body>...</>
1007 </e:Envelope>
    
```

1008 The API, such as *tas3\_get\_tas3\_status()* for creating and inspecting TAS<sup>3</sup> Status Header is described in  
 1009 section 3.1.13 "SOAP Fault and Status Generation and Inspection".

## 1011 2.14.2 TAS<sup>3</sup> Status Codes

1012 The code XML attribute may contain any of the ID-WSF defined status codes, see [[SOAPBinding2](#)]  
 1013 Table 2 on pp.12-13, including the special value "OK" to indicate success. It may also contain any appli-  
 1014 cation specific status indications, provided that they are qualified to their own namespace using URN or  
 1015 URL constructs. Finally it may contain any of the following TAS<sup>3</sup> defined status codes:

1016 **urn:tas3:status:deny** Operation denied by authorization layer

1017 **urn:tas3:status:notapplicable** Operation not applicable from authorization perspective

1018 **urn:tas3:status:indeterminate** Operation's status can not be determined by the authorization  
 1019 layer

1020 **urn:tas3:status:nosig** Operation denied due to required signature missing.

1021 **urn:tas3:status:badsig** Operation denied due to signature validation problem.

1022 **urn:tas3:status:badcond** Expiry time or audience restriction did not validate.

1023

## 1024 2.14.3 TAS<sup>3</sup> Control and Reporting Points

1025 The status messages can emanate from several parts in TAS<sup>3</sup> security layer, or even from points inside  
 1026 the application. To assist in determining where errors originate, the `<tas3:Status>` element carries a  
 1027 `ctlpt` XML attribute, whose value is a URI identifying the origin of the error. While application can  
 1028 define a number of additional URIs, the TAS<sup>3</sup> architecture defines the following:

1029 **urn:tas3:ctlpt:pep:rq:out** Request Out PEP (callout 1)

1030 **urn:tas3:ctlpt:pep:rq:in** Request In PEP (callout 2)

1031 **urn:tas3:ctlpt:pep:rs:out** Response Out PEP (callout 3)

1032 **urn:tas3:ctlpt:pep:rs:in** Response In PEP (callout 4)

1033 **urn:tas3:ctlpt:app** Application. In this case application can also define its own URIs.

1034

## 1035 2.15 Registration of Business Process Models

1036 The attribute needs and participants of the business process model are declared using CARML declara-  
 1037 tion. Each business process model is assigned a service typi URI, which is used by the SPs that implement  
 1038 the business process model to register themselves in the discovery.

1039

## 1040 3 The Official TAS<sup>3</sup> API (normative, but non-exclusive)

1041 Although wire-interoperability is the main goal of the TAS<sup>3</sup> project, we recognize that interoperability  
 1042 at software interface level, i.e. interchangeable implementations of an API, is valuable as well. Stan-  
 1043 dardization of APIs, in addition to wire protocols, helps to promote building a culture and community of  
 1044 programmers catering for the TAS<sup>3</sup> platform. Such community fosters adoption through mutual self help  
 1045 and shared knowledge base. Supporting full constellation of APIs for all programming languages and  
 1046 platforms is fairly expensive business, but is necessary to address the present fragmented market.

1047 The TAS<sup>3</sup> API described herein is meant to have multiple implementations. Each implementation  
 1048 provides

- 1049 • The interface files described herein, such as `tas3.h`
- 1050 • Libraries or implementation files that provide the symbols described by the interface files. In as far  
 1051 as possible, these will be called `libtas3.so`, `libtas3.dll`, or other appropriate and similar name.  
 1052 However a concrete implementation may choose to incorporate the TAS<sup>3</sup> API interface in its own  
 1053 library, or may require its own library to be included in addition to the `libtas3.*` library. Such  
 1054 additional requirements shall be conspicuously described in the implementation documentation.

1055 The official TAS<sup>3</sup> API is not meant to exclude other wire-protocol compatible implementations of TAS<sup>3</sup>.  
 1056 Thus, while there is only one official API, other APIs can be equally TAS<sup>3</sup> compatible on the wire.

1057 The particular API in use is chosen by the programmer by including the appropriate header file or in-  
 1058 terface description. The particular API implementation in use is chosen by the system administrator or  
 1059 the programmer by linking against a particular library providing the TAS<sup>3</sup> binary interface, or by dy-  
 1060 namically loading a module implementing the said binary interface. This leaves great implementation  
 1061 flexibility while accurately describing the TAS<sup>3</sup> interface and implementation at source code (API) and  
 1062 binary (ABI) level.

### 1064 3.1 Language Independent Description of the API

1065 Since all language specific bindings, by-and-large, share the same semantics, the functions and meth-  
 1066 ods are first described generically, using pseudocode if needed. Each language binding takes the same  
 1067 parameters and behaves in the way that API would naturally work, *mutantis mudandis*, for that language.<sup>1</sup>

1068 The five essential APIs are

1069 *tas3\_sso()* SSO (with optional application independent authorization)

1070 *tas3\_az()* Application Dependent Authorization

1071 *tas3\_call()* Web Services Client: call a web service and validate response

1072 *tas3\_wsp\_validate()* Validate that web service request can be processed

1073 *tas3\_wsp\_decorate()* Create a web service response

1074

#### 1075 3.1.1 Single Sign On (SSO) Alternatives

1076 The TAS<sup>3</sup> SSO API's primary aim is supporting SAML 2.0 SSO (and SLO) with attribute and bootstrap  
 1077 passing. Not all COTS SAML 2.0 SP APIs (or IdPs) are capable of this out of the box. Thus being SAML  
 1078 2.0 compatible is a prerequisite, but additional properties, such as specific functions, session level attribute  
 1079 pool, and bootstrap cache, must be satisfied as well to be TAS<sup>3</sup> API compliant. The TAS<sup>3</sup> SSO API is  
 1080 likely to support in future (as of 2009) in a transparent way InfoCard specification [[CardSpace](#)], and may  
 1081 be able to support other SSO specifications as well.

<sup>1</sup>Some procedural bias is evident, even in "object oriented" language bindings. This is due to least-common-denominator syndrome, i.e. desire to have same API for all programming languages.

1082 Some alternatives for supporting SSO:

- 1083 • `mod_auth_saml` and (Apache) subprocess environment provides a complete solution for SSO layer  
1084 if using Apache `httpd` or compatible web server. In such case the SSO is handled without any  
1085 programming simply by editing `httpd.conf` (and in some cases `zxid.conf`). The `mod_auth_saml`  
1086 configuration directives are the same as in `zxid.org` and they are introduced to `httpd.conf` using  
1087 `ZXIDConf` directives.
- 1088 • `tas3_sso()` API as complete solution. `tas3_sso()` API implements a state machine that the calling  
1089 application must crank by making repeated calls (one per HTTP request until SSO completes). This  
1090 approach has a benefit of isolating the calling application from protocol flow specifics and allows  
1091 the API to support multiple SSO protocols in a transparent manner.
- 1092 • `tas3_sso_servlet.class`: Java servlet that can be configured to Tomcat or other servlet container to  
1093 implement SSO for payload servlets. Internally the SSO servlet calls `tas3_simple()`;
- 1094 • **Deprecated Alternative**: by steps approach using medium level APIs (deprecated because the logic  
1095 of the specific SSO protocol flow would be hardwired into the calling application)  
1096

1097 **3.1.2 SSO: `ret = tas3_sso(conf, qs, auto_flags)`**

1098 The `tas3_sso()` API is essentially a Single Sign-On protocol state machine. Unless the application  
1099 already has a valid active session established, it should call `tas3_sso()` upon every HTTP request, passing  
1100 in the query string or form submission part as the `qs` argument. The argument is a string and must be  
1101 formatted as a query string. The `tas3_sso()` then returns a string which the calling application needs  
1102 to interpret to decide what to do next. Possible actions include performing HTTP redirect, sending the  
1103 returned string as HTTP response, or completing a successful single sign on.

1104 When Single Sign-On is completed, the `tas3_sso()` establishes a session object for holding received  
1105 attributes and bootstrap EPRs. These can be accessed from the session either by the calling application,  
1106 or by other TAS<sup>3</sup> API functions such as `tas3_az()` and `tas3_call()`. The `tas3_sso()` may incorporate a  
1107 configurable frontend policy enforcement point. Such configuration is implementation dependent.

1108 There are many options. Most of these have sensible default values or can be specified in a configuration  
1109 file. The first parameter either is a configuration object, or a configuration string that modifies or adds to the  
1110 default configuration. Some aspects of operation of `tas3_sso()` are affected by the `auto_flags` parameter.

Table 3.1: `tas3_sso()` configuration options that all implementations MUST support

Option	Description
PATH	Path of configuration directory, which contains the configuration file and may contain other implementation dependent information.
URL	Base URL from which the EntityID is formed.

Table 3.2: *tas3\_sso()* AUTO flags

Dec	Hex	Symbol	Description
1	0x01	TAS3_AUTO_EXIT	Call <i>exit(2)</i> , 0=return "n", even if auto CGI
2	0x02	TAS3_AUTO_REDIR	Automatic. handle redirects, assume CGI (calls <i>exit(2)</i> )
4	0x04	TAS3_AUTO_SOAPC	SOAP response handling, content gen
8	0x08	TAS3_AUTO_SOAPH	SOAP response handling, header gen
16	0x10	TAS3_AUTO_METAC	Metadata response handling, content gen
32	0x20	TAS3_AUTO_METAH	Metadata response handling, header gen
64	0x40	TAS3_AUTO_LOGINC	IdP select / Login page handling, content gen
128	0x80	TAS3_AUTO_LOGINH	IdP select / Login page handling, header gen
256	0x100	TAS3_AUTO_MGMTC	Management page handling, content gen
512	0x200	TAS3_AUTO_MGMTH	Management page handling, header gen
1024	0x400	TAS3_AUTO_FORMF	In IdP list and mgmt screen, generate form fields
2048	0x800	TAS3_AUTO_FORMT	In IdP list & mgmt screen, wrap in <form> tag.
4095	0xffff	TAS3_AUTO_ALL	Enable all automatic CGI behaviour.
4096	0x1000	TAS3_AUTO_DEBUG	Enable debugging output to stderr.
8192	0x2000	TAS3_AUTO_OFMTQ	Output Format Query String
16384	0x4000	TAS3_AUTO_OFMTJ	Output Format JSON

### 1111 Example Usage

```

1112 01 res = tas3_sso(conf, request['QUERY_STRING'], 0x1800);
1113 02 switch (substr(res, 0, 1)) {
1114 03 case 'L': header(res); return 0; # Redirect
1115 04 case 'n': return 0; # already handled
1116 05 case 'b': return my_send_metadata();
1117 06 case 'e': return my_render_idp_selection_screen();
1118 07 case 'd': return my_start_session_and_render_protected_content();
1119 08 default: error_log("Unknown tas3_sso() res(%s)", res); return 0;
1120 09 }
    
```

### 1121 Return values

1122 The return value starts by an action letter and may be followed by data that is relevant for the action.

1123 **L** Redirection request (L as in Location header). The full contents of the res is the redirection request,  
 1124 ready to be printed to stdout of a CGI. If you want to handle the redirection some other way, you  
 1125 can parse the string to extract the URL and do your thing. This res is only returned if you did not  
 1126 set TAS3\_AUTO\_REDIR.

1127 Example:

```
1128 Location: https://sp1.zxidsp.org:8443/zxid?o=C
```

1129 **C** Content with Content-type header. The res is ready to be printed to the stdout of a CGI, but if you want  
 1130 to handle it some other way, you can parse the res to extract the header and the actual body.

1131 Example:

```

1132 CONTENT-TYPE: text/html
1133
1134 <title>Login page</title>
1135 ...
    
```

1136 Example (metadata):

```
1137     CONTENT-TYPE: text/xml
1138
1139     <m:EntityDescriptor>
1140     ...
```

1141 **Less than (" $<$ ")** Content without headers. This could be HTML content for login page or metadata  
 1142 XML. To know which (and set content type correctly), you would have to parse the content.  
 1143 This res format is only applicable if you did not specify TAS3\_AUTO\_CTYPE (but did specify  
 1144 TAS3\_AUTO\_CONTENT).

1145 **n** Do nothing. The operation was somehow handled internally but the *exit(2)* was not called (e.g. TAS3\_AUTO\_SOAP  
 1146 was NOT specified). The application should NOT attempt generating any output.

1147 **b** Indication that the application should send SP metadata to the client. This res is only returned if you  
 1148 did not set TAS3\_AUTO\_META.

1149 **c** Indication that the application should send SP CARMML declaration to the client. This res is only re-  
 1150 turned if you did not set TAS3\_AUTO\_META.

1151 **e** Indication that the application should display the IdP selection page. This res is only returned if you did  
 1152 not set TAS3\_AUTO\_CONTENT.

1153 **d** Indication that SSO has been completed or that there was an existing valid session in place. The res is  
 1154 an LDIF entry containing attributes that describe the SSO or session.

```
1155     dn: idpnid=Pa45XAs2332SDS2asFs,affid=https://idp.demo.com/idp.xml
1156     objectclass: zxidsession
1157     affidavit: https://idp.demo.com/idp.xml
1158     idpnid: Pa45XAs2332SDS2asFs
1159     authnctxlevel: password
1160     sesid: S12aF3Xi4A
1161     cn: Joe Doe
```

1162 Usually your application would parse the attributes and then render its application specific content.

1163 **z** Authorization failure. Application MUST NOT display protected content. Instead, it should offer  
 1164 user interface where the user can understand what happened and possibly gain the extra credentials  
 1165 needed.

1166 **Asterisk ("\*")** Although any unknown letter should be interpreted as an error, we follow convention of  
 1167 prefixing errors with an asterisk ("\*").  
 1168

### 1169 3.1.3 Authorization: decision = *tas3\_az(conf, qs, ses)*

1170 Implicit application independent authorization steps are performed in *tas3\_sso()* SSO, *tas3\_call()* Ser-  
 1171 vice Requester, *tas3\_wsp\_validate()*, and *tas3\_wsp\_decorate()* APIs. To activate them, you need to supply  
 1172 appropriate configuration options. Specifics of this configuration are implementation dependent.

1173 The *tas3\_az()* function is the main work horse for requesting authorization decisions from the PDPs.  
 1174 It allows programmer to make Application Dependent authorization calls, supplying some or all of the  
 1175 attributes needed in a XACML request. *tas3\_az()* can also use attributes from the session, if configured.  
 1176 Specifics of this configuration are implementation dependent.

1177 **conf** the configuration string or object

1178 **qs** if supplied, any CGI variables are imported to session environment as attributes according to configu-  
 1179 ration. Format is CGI Query String.

1180 **ses** attributes are obtained from the session, if supplied (see also CGI). Session ID can be supplied as a  
 1181 string or a session object can be passed.

1182 **return** 0 if deny (for any reason, e.g. indeterminate), or string if permit

### 1183 **Example Pseudocode**

```

1184 cf = tas3_new_conf();
1185 ses = tas3_alloc_ses(cf);
1186 ret = tas3_simple_cf_ses(cf, 0, $QUERY_STRING, ses, 0, 0x1800);
1187 if (ret =~ /^d/) {
1188     perr "SSO ok, now checking authorization";
1189     if (tas3_az_cf_ses(cf, "Action=SHOW&BusinessProcess=register:emp", ses))
1190         perr "Permit, add code to deliver application content";
1191     else
1192         perr "Deny, send back an error";
1193 }
1194
    
```

### 1195 **3.1.4 Web Service Call: `ret_soap = tas3_call(cf, ses, svctype, url, di_opt,`** 1196 **`az_cred, req_soap)`**

1197 `tas3_call()` first checks if `req_soap` string is already a SOAP envelope. If not, it will supply miss-  
 1198 ing `<Envelope>`, `<Header>`, and `<Body>` elements. You still need to pass something in `req_soap` as  
 1199 `tas3_call()` can not guess the contents of the `<Body>` - it can only add the wrapping. The idea is that the  
 1200 programmer can concentrate on application layer and the `tas3_call()` will supply the rest automatically. If,  
 1201 however, the programmer wishes to pass some SOAP headers, he can do so by passing the entire envel-  
 1202 ope. Even if entire envelope is passed, `tas3_call()` will add TAS<sup>3</sup> specific headers and signatures to this  
 1203 envelope.

1204 Similarly on return, `tas3_call()` will check all TAS<sup>3</sup> relevant SOAP headers and signatures, but will  
 1205 still return the entire SOAP envelope as a string so that the application layer can, if it wants, look at the  
 1206 headers.

1207 Next, `tas3_call()` will attempt to locate an EPR for the service type. This may already be in the session  
 1208 cache, or a discovery step may be performed. If discovery is needed it will be automatically made. The  
 1209 discovery can be constrained using `url` and `di_opt` parameters. For example, if there is a predetermined  
 1210 (list of) service provider(s), the `url` parameter can be used to force the choice. Discovery may still be  
 1211 done to obtain credentials needed for the call, but the discovery result will be constrained to match the  
 1212 supplied `url`. See section `tas3_get_epr()` for description of explicit discovery.

1213 Before actual SOAP call, `tas3_call()` may contact a PDP to authorize the outbound call. This corre-  
 1214 sponds to application independent *Requester Out PEP* and is configurable: you can disable it if you prefer  
 1215 to make an explicit application dependent call to `tas3_az()`. The attributes for the XACML request are  
 1216 mainly derived from the session, but additional attributes can be supplied with `az_cred` parameter, which  
 1217 has query string format. Functioning of the authorization step can be controlled using configuration, which  
 1218 is implementation dependent.

1219 Then `tas3_call()` augments the XML data structure with Liberty ID-WSF mandated headers. It will  
 1220 look at the security mechanism and token specified in the EPR and perform appropriate steps to create  
 1221 WS-Security header and apply signature as needed.

1222 Next `tas3_call()`, using its built-in http client, opens TCP connection to the web service provider and  
 1223 sends the SOAP envelope using HTTP protocol. It then waits for the HTTP response, blocking until the  
 1224 response is received.

1225 After executing the SOAP call and verifying any returned TAS<sup>3</sup> relevant headers and signatures, *tas3\_call()*  
 1226 may contact a PDP to authorize receiving data, and to pass on any obligations that were received. This  
 1227 corresponds to application independent *Requester In PEP* and is configurable: you can disable it if you  
 1228 prefer to make explicit application dependent call to *tas3\_az()*. The contents of the XACML request are  
 1229 determined based on the response, session, *az\_cred* parameter, which is shared for both Responder Out  
 1230 and Responder In PDP calls, and configuration, which is implementation dependent.

1231 **cf** Configuration object, see *tas3\_new\_conf\_to\_cf()*

1232 **ses** Session object, used to locate EPRs, see *tas3\_new\_ses()*

1233 **svctype** Service type and namespace URN that is applicable to the body. Passed as a string.

1234 **url** (Optional) If provided, this argument has to match either the ProviderID, EntityID, or actual service  
 1235 endpoint URL.

1236 **di\_opt** (Optional) Additional discovery options for selecting the service, query string format

1237 **az\_cred** (Optional) Additional authorization credentials or attributes, query string format. These creden-  
 1238 tials will be populated to the session's attribute pool in addition to the ones obtained from SSO  
 1239 and other sources. Then a PDP is called to get an authorization decision (as well as obligations  
 1240 we pledge to support). This implements generalized (application independent) Requester Out and  
 1241 Requester In PEPs. To implement application dependent PEP features you should call *tas3\_az()*  
 1242 directly.

1243 **req\_soap** string used as SOAP body or as SOAP envelope template.

1244 **return** SOAP envelope as a string.

### 1245 Example

```

1246 01 env = tas3_callf(cf, ses, 0,0,0, "urn:hrxml:idhrxml",
1247 02     "<idhrxml:Modify>"
1248 03     "<idhrxml:ModifyItem>"
1249 04     "<idhrxml:Select>%s</idhrxml:Select>"
1250 05     "<idhrxml:NewData>%s</idhrxml:NewData>"
1251 06     "</idhrxml:ModifyItem>"
1252 07     "</idhrxml:Modify>", cgi.select, cgi.data);
1253 08 if (env) {
1254 09     xml = xml_parse(env);
1255 10     if (xml->Status->code == "OK") {
1256 11         INFO("Data is " + xml->Data);
1257 12     } else {
1258 13         ERR("Web service error " + xml->Status->code);
1259 14     }
1260 15 } else {
1261 16     ERR("HTTP failure");
1262 17 }
    
```

1263 As can be seen, the paradigm is to supply the payload data as a string. Although it could be supplied  
 1264 as a data structure, constructed with many constructors, our experience has shown that string representa-  
 1265 tion is most intuitive and self documenting for most programmers. Despite abandoning the constructor  
 1266 approach, all relevant syntax and schema checks are internally done by simply parsing the string and then  
 1267 reserializing it before sending to the wire. This tends to be necessary anyway due to signature generation.

1268

### 1269 3.1.5 Requester out: req\_decor\_soap = tas3\_wsc\_prepare\_call(cf, ses, svc- 1270 type, az\_cred, req\_soap)

1271 This API function decorates a request envelope with necessary ID-WSF SOAP headers and signs it, but  
1272 does not send the envelope. This API is used as a building block in *tas3\_call()*, which see. Usually you  
1273 should use *tas3\_call()* instead of this API function.

### 1275 3.1.6 Requester in: status = tas3\_wsc\_valid\_resp(cf, ses, az\_cred, res\_decor\_soap)

1276 This API function validates response envelope checking necessary ID-WSF SOAP headers and signa-  
1277 ture. This API is used as a building block in *tas3\_call()*, which see. Usually you should use *tas3\_call()*  
1278 instead of this API function.

1279 *tas3\_wsc\_prepare\_call()* and *tas3\_wsc\_valid\_resp()* work together as follows:

```
1280 01 req_soap = tas3_wsc_prepare_call(cf, ses, svctype,
1281 02                               url, di_opt, az_cred,
1282 03                               "<idhrxml:Modify>...</>");
1283 04 resp_soap = your_http_post_client(url, req_soap);
1284 05 if (tas3_wsc_valid_resp(cf, ses, az_cred, resp_soap)) {
1285 06     xml = xml_parse(resp_soap);
1286 07     INFO("Data is " + xml->Data);
1287 08 } else
1288 09     ERR("HTTP failure");
1289
```

### 1290 3.1.7 Responder in: tgtnid = tas3\_wsp\_validate(cf, ses, az\_cred, soap\_req)

1291 Validate SOAP request (envelope), specified by the string *soap\_req*. Service Responder should call  
1292 this function to validate an inbound, received, TAS<sup>3</sup> request. This will

- 1293 • verify signatures
- 1294 • determine trust
- 1295 • populate to WSP's session any credentials found in the request
- 1296 • possibly perform an application independent *Responder In PEP* authorization, calling a PDP behind  
1297 the scenes using *tas3\_az()*.

1298 After *tas3\_wsp\_validate()*, the application needs to, in application dependent way, extract from the  
1299 response the application payload and process it. However, this is much simplified as there is no need to  
1300 perform any further verification.

1301 If the string *soap\_req* starts by "<e:Envelope", then it should be a complete SOAP envelope including  
1302 <e:Header> (and <e:Body>) parts.

1303 **cf** TAS<sup>3</sup> configuration object, see *tas3\_new\_conf()*

1304 **ses** Session object that contains the EPR cache, see *tas3\_new\_ses()*

1305 **az\_cred** (Optional) Additional authorization credentials or attributes, query string format. These creden-  
1306 tials will be populated to the attribute pool in addition to the ones obtained from token and other  
1307 sources. Then a PDP is called to get an authorization decision (matching obligations we support  
1308 to those in the request, and obligations pledged by caller to those we insist on). This implements  
1309 generalized (application independent) *Responder In PEP*. To implement application dependent PEP  
1310 features you should call *tas3\_az()* directly.

1311 **soap\_req** Entire SOAP envelope as a string

1312 **return** idpnid, as a string, of the target identity of the request (rest of the information is populated to the  
1313 session object, from where it can be retrieved).

1314

### 1315 3.1.8 Responder out: soap = *tas3\_wsp\_decorate(cf, ses, az\_cred, soap\_resp)*

1316 Add ID-WSF (and TAS3) specific headers and signatures to web service response. Simple and intuitive  
1317 specification of XML as string: no need to build complex data structures.

1318 Service responder should prepare application layer of the response and then call this function to decorate  
1319 the response with TAS3 specifics, and to wrap it in SOAP envelope. This will

- 1320 • add correlation headers
- 1321 • possibly perform an application independent *Responder Out PEP* authorization step, calling a PDP  
1322 behind the scenes using *tas3\_az()*.
- 1323 • apply signature

1324 If the string starts by "<e:Envelope", then string should be a complete SOAP envelope including  
1325 <e:Header> and <e:Body> parts. This allows caller to specify custom SOAP headers, in addition to  
1326 the ones that the underlying *zxid\_wsc\_call()* will add. Usually the payload service will be passed as the  
1327 contents of the body. If the string starts by "<e:Body", then the <e:Envelope> and <e:Header> are  
1328 automatically added. If the string does not start by "<e:Envelope" or "<e:Body"<sup>2</sup>, then it is assumed to  
1329 be the payload content of the <e:Body> and the rest of the SOAP envelope is added.

1330 **cf** TAS<sup>3</sup> configuration object, see *tas3\_new\_conf()*

1331 **ses** Session object that contains the EPR cache

1332 **az\_cred** (Optional) Additional authorization credentials or attributes, query string format. These cre-  
1333 dentials will be populated to the attribute pool in addition to the ones obtained from token and  
1334 other sources. Then a PDP is called to get an authorization decision (generating obligations). This  
1335 implements generalized (application independent) *Responder Out PEP*. To implement application  
1336 dependent PEP features you should call *tas3\_az()* directly.

1337 **soap\_resp** XML payload as a string

1338 **return** SOAP Envelope of the response, as a string, ready to be sent as HTTP response.

1339

### 1340 3.1.9 Explicit Discovery: epr = *tas3\_get\_epr(cf, ses, svc, url, di\_opt, act, n)*

1341 N.B. This function is automatically called by *tas3\_call()* so making an explicit call is seldom  
1342 needed. You may consider making such call if you need to know which EPR is actually found  
1343 and you want to query some properties of the EPR. You can then pass the URL, as found using  
1344 *tas3\_get\_epr\_url()*, as an argument to *tas3\_call()* to constrain the call to use a specific EPR.

1345 First search the epr cache, and if there is a cache miss, go discover an EPR over the net. This is the  
1346 main work horse for WSCs wishing to call WSPs via EPR.

1347 **cf** TAS<sup>3</sup> configuration object, also used for memory allocation

1348 **ses** Session object in whose EPR cache the file will be searched

---

<sup>2</sup>Be careful to use the "e:" as namespace prefix if you want e:Envelope or e:Body to be detected.

1349 **svc** Service type (usually a URN). String.

1350 **url** (Optional) If provided, this argument has to match either the ProviderID, EntityID, or actual service  
1351 endpoint URL. String.

1352 **di\_opt** (Optional) Additional discovery options for selecting the service, query string format.

1353 **act** (Optional) The action, or method, that must be invocable on the service. String.

1354 **n** Which matching instance is returned. 1 means first. Integer.

1355 **return** EPR data structure on success, null on failure (no discovery EPR in cache, or not found by the  
1356 discovery service).

1357

### 1358 **3.1.10 url = tas3\_get\_epr\_url(cf, epr)**

1368 Returns the <a:Address> field of an EPR as a string. This is the endpoint URL.

### 1361 **3.1.11 entityid = tas3\_get\_epr\_entid(cf, epr)**

1368 Returns the <di:ProviderID> field of an EPR as a string. This is same as SAML2 EntityID.

### 1364 **3.1.12 a7n = tas3\_get\_epr\_a7n(cf, epr)**

1368 Returns assertion from EPR <sec:Token> field as a string.

### 1367 **3.1.13 SOAP Fault and Status Generation and Inspection**

1368 Error reporting using SOAP faults and TAS<sup>3</sup> status header is discussed in section 2.13 "Uniform Appli-  
1369 cation Status and Error Reporting"

1370 tas3\_status\* tas3\_mk\_tas3\_status(tas3\_conf\* cf, const char\* ctlpt, const char\* sc1, const char\* sc2,  
1371 const char\* msg, const char\* ref); struct zx\_e\_Fault\_s\* tas3\_mk\_fault(tas3\_conf\* cf, const char\* fa, const  
1372 char\* fc, const char\* fs, const char\* sc1, const char\* sc2, const char\* msg, const char\* ref);

1373 void tas3\_set\_fault(tas3\_conf\* cf, tas3\_ses\* ses, struct zx\_e\_Fault\_s\* ft); struct zx\_e\_Fault\_s\* tas3\_get\_fault(tas3\_conf\*  
1374 cf, tas3\_ses\* ses);

1375 char\* tas3\_get\_tas3\_fault\_sc1(tas3\_conf\* cf, struct zx\_e\_Fault\_s\* ft); char\* tas3\_get\_tas3\_fault\_sc2(tas3\_conf\*  
1376 cf, struct zx\_e\_Fault\_s\* ft); char\* tas3\_get\_tas3\_fault\_comment(tas3\_conf\* cf, struct zx\_e\_Fault\_s\* ft);  
1377 char\* tas3\_get\_tas3\_fault\_ref(tas3\_conf\* cf, struct zx\_e\_Fault\_s\* ft); char\* tas3\_get\_tas3\_fault\_actor(tas3\_conf\*  
1378 cf, struct zx\_e\_Fault\_s\* ft);

1379 void tas3\_set\_tas3\_status(tas3\_conf\* cf, tas3\_ses\* ses, tas3\_status\* status); tas3\_status\* tas3\_get\_tas3\_status(tas3\_conf\*  
1380 cf, tas3\_ses\* ses);

1381 char\* tas3\_get\_tas3\_status\_sc1(tas3\_conf\* cf, tas3\_status\* st); char\* tas3\_get\_tas3\_status\_sc2(tas3\_conf\*  
1382 cf, tas3\_status\* st); char\* tas3\_get\_tas3\_status\_comment(tas3\_conf\* cf, tas3\_status\* st); char\* tas3\_get\_tas3\_status\_ref(ta  
1383 cf, tas3\_status\* st); char\* tas3\_get\_tas3\_status\_ctlpt(tas3\_conf\* cf, tas3\_status\* st);

1384

## 1385 3.2 Java Binding

1386 Before you start using the SSO API, you should consider using the TAS<sup>3</sup> SSO servlet. `tas3_sso_servlet.class`  
 1387 can be configured to Tomcat or other servlet container to implement SSO for payload servlets. Internally  
 1388 the SSO servlet calls `tas3_sso()`.

1389 Similar module is planned (as of 2009) for Responder implementation. The pushable filter module  
 1390 for servlet environments (e.g. Tomcat) will wrap `tas3.wsp_validate()` and `tas3.wsp_decorate()`. The filter  
 1391 module allows some web services to be TAS<sup>3</sup> enabled without modification to the application code.

### 1393 3.2.1 Interface and Initialization

1394 This binding is implemented as `tas3java.class` and `libtas3jni.so` (`libtas3jni.jnilib` on MacOS  
 1395 X, `libtas3jni.dll` on Windows) module.

1396 Typically you need to include in your Java servlet or program something like

```
1397 01 import tas3java.*;
1398 02 static tas3.tas3_conf cf;
1399 03 static {
1400 04     System.loadLibrary("tas3jni");
1401 05     cf = tas3.new_conf_to_cf("PATH=/var/tas3/");
1402 06 }
```

1403 This will bring in the functionality of the TAS<sup>3</sup> Java binding and cause the JNI library implementing  
 1404 this functionality to be loaded. It will also create a configuration object that the other parts of a servlet can  
 1405 share.

1406 The Java binding replaces the "tas3\_" prefix in function names with the class prefix "tas3.", for example  
 1407 `tas3_sso()` becomes `tas3.sso()` and `tas3_az()` becomes `tas3.az()`.

1408 The TAS<sup>3</sup> Java interface is defined as follows

```
1409 package tas3;
1410
1411 public interface tas3 {
1412     public static tas3_conf new_conf_to_cf(String conf);
1413     public static tas3_ses new_ses(tas3_conf cf);
1414     public static tas3_ses fetch_ses(tas3_conf cf, String sid);
1415     public static String sso_cf(tas3_conf cf, int qs_len, String qs,
1416         p_int res_len, int auto_flags);
1417     public static int get_ses(tas3_conf cf, tas3_ses ses, String sid);
1418     public static int az_cf_ses(tas3_conf cf, String qs, tas3_ses ses);
1419     public static int az_cf(tas3_conf cf, String qs, String sid);
1420     public static int az(String conf, String qs, String sid);
1421
1422     public static String wsp_validate(tas3_conf cf, tas3_ses ses,
1423         String az_cred, String enve);
1424     public static String wsp_decorate(tas3_conf cf, tas3_ses ses,
1425         String az_cred, String enve);
1426     public static String call(tas3_conf cf, tas3_ses ses,
1427         String svctype, String url, String di_opt,
1428         String az_cred, String enve);
1429     public static tas3_epr get_epr(tas3_conf cf, tas3_ses ses,
1430         String svc, String url, String di_opt,
1431         String action, int n);
```

```

1432     public static String get_epr_url(tas3_conf cf, tas3_epr epr);
1433     public static String get_epr_entid(tas3_conf cf, tas3_epr epr);
1434     public static String get_epr_a7n(tas3_conf cf, tas3_epr epr);
1435 }
1436
1437
    
```

### 1438 **3.2.2 Initialize: cf = tas3.new\_conf\_to\_cf(conf)**

1439 Create a new TAS3 configuration object given configuration string and possibly configuration file. Usually a configuration object is generated and passed around to different API calls to avoid reparsing the configuration at each API call.

1442 **conf** Configuration string

1443 **return** Configuration object

1444

### 1445 **3.2.3 New session: ses = tas3.new\_ses(cf)**

1446 Create a new TAS3 session object. Usually a session object is created just before calling *zxidjni.wsp\_validate()*.

1447 **cf** Configuration object, see *tas3.new\_conf\_to\_cf()*

1448 **return** Session object

1449

### 1450 **3.2.4 SSO: ret = tas3.sso\_cf\_ses(cf, qs\_len, qs, ses, null, auto\_flags)**

1451 **cf** Configuration object, see *tas3.new\_conf\_to\_cf()*

1452 **qs\_len** Length of the query string. -1 = use *strlen()*

1453 **qs** Query string (or POST content)

1454 **ses** Session object, see *tas3.new\_ses()*. Session object is modified.

1455 **res\_len** Result parameter. Must always pass *null* as result parameters are not supported in the Java binding.

1456

1457 **auto\_flags** Automation flags

1458 **return** String representing protocol action or SSO attributes

1459

### 1460 **3.2.5 Authorization: decision = tas3.az\_cf\_ses(cf, qs, ses)**

1461 **cf** the configuration object, see *tas3.new\_conf\_to\_cf()*

1462 **qs** additional attributes that are passed to PDP

1463 **ses** session object, from which most attributes come

1464 **return** 0 on deny (for any reason, e.g. indeterminate), or non-null if permit.

1465

1466 **3.2.6 WSC: resp\_soap = *tas3.call(cf, ses, svctype, url, di\_opt, az\_cred, req\_soap)***

1467 **cf** Configuration object, see *tas3.new\_conf\_to\_cf()*

1468 **ses** Session object, used to locate EPRs, see *tas3.new\_ses()*

1469 **svctype** Service type and namespace URN that is applicable to the body. Passed as a string.

1470 **url** (Optional) If provided, this argument has to match either the ProviderID, EntityID, or actual service  
1471 endpoint URL.

1472 **di\_opt** (Optional) Additional discovery options for selecting the service, query string format

1473 **az\_cred** (Optional) Additional authorization credentials or attributes, query string format.

1474 **req\_soap** string used as SOAP body or as SOAP envelope template.

1475 **return** SOAP envelope as a string

1476

1477 **3.2.7 WSP: tgtid = *tas3.wsp\_validate(cf, ses, az\_cred, soap\_req)***

1478 **cf** TAS<sup>3</sup> configuration object, see *tas3.new\_conf\_to\_cf()*

1479 **ses** Session object that contains the EPR cache, see *tas3.new\_ses()*

1480 **az\_cred** (Optional) Additional authorization credentials or attributes, query string format.

1481 **soap\_req** Entire SOAP envelope as a string

1482 **return** idpnid, as a string, of the target identity of the request (rest of the information is populated to the  
1483 session object, from where it can be retrieved).

1484

1485 **3.2.8 WSP: soap = *tas3.wsp\_decorate(cf, ses, az\_cred, soap\_resp)***

1486 **cf** TAS<sup>3</sup> configuration object, see *tas3.new\_conf\_to\_cf()*

1487 **ses** Session object that contains the EPR cache

1488 **az\_cred** (Optional) Additional authorization credentials or attributes, query string format.

1489 **soap\_resp** XML payload, as a string

1490 **return** SOAP Envelope of the response, as a string, ready to be sent as HTTP response.

1491

1492 **3.2.9 Explicit Discovery: epr = *tas3.get\_epr(cf, ses, svc, url, di\_opt, act, n)***

1493 First search epr cache, and if miss, go discover an EPR over the net. This is the main work horse for  
1494 WSCs wishing to call WSPs via EPR.

1495 **cf** TAS<sup>3</sup> configuration object, also used for memory allocation

1496 **ses** Session object in whose EPR cache the file will be searched

1497 **svc** Service type (usually a URN)

1498 **url** (Optional) If provided, this argument has to match either the ProviderID, EntityID, or actual service  
1499 endpoint URL.

1500 **di\_opt** (Optional) Additional discovery options for selecting the service, query string format  
 1501 **act** (Optional) The action, or method, that must be invocable on the service  
 1502 **n** Which matching instance is returned. 1 means first  
 1503 **return** EPR data structure on success, 0 on failure (no discovery EPR in cache, or not found by the  
 1504 discovery service).  
 1505

### 1506 **3.2.10 url = tas3.get\_epr\_url(cf, epr)**

1507 **cf** TAS<sup>3</sup> configuration object, also used for memory allocation  
 1508 **epr** An EPR object, such as obtained from *tas3\_get\_epr()*  
 1509 **return** The <a:Address> field of an EPR as a string. This is the endpoint URL.  
 1510

### 1511 **3.2.11 entityid = tas3.get\_epr\_entid(cf, epr)**

1512 **cf** TAS<sup>3</sup> configuration object, also used for memory allocation  
 1513 **epr** An EPR object, such as obtained from *tas3\_get\_epr()*  
 1514 **return** The <di:ProviderID> field of an EPR as a string. This is same as SAML2 EntityID.  
 1515

### 1516 **3.2.12 a7n = tas3.get\_epr\_a7n(cf, epr)**

1517 **cf** TAS<sup>3</sup> configuration object, also used for memory allocation  
 1518 **epr** An EPR object, such as obtained from *tas3\_get\_epr()*  
 1519 **return** Assertion from EPR <sec:Token> field as a string.  
 1520

### 1521 **3.2.13 Available Implementations (Non-normative)**

1522 This binding is implemented using Java Native Interface calls to *zxid.org* C library by *zxidjni* module.  
 1523 Other implementations are welcome.

1524

## 1525 3.3 PHP Binding

1526 Using TAS<sup>3</sup> PHP APIs requires first loading the TAS<sup>3</sup> module and creating a configuration object.  
 1527 These are typically accomplished from PHP initialization. You may consider creating `tas3.ini` file:

```
1528 dl("php_tas3.so");
1529 $cf = tas3_new_conf_to_cf("PATH=/var/tas3/");
1530
```

### 1531 3.3.1 Application Level Integration

1532 It should be noted that many PHP applications run inside Apache httpd and therefore can accomplish  
 1533 SSO using `mod_auth_saml` approach without any programming. Especially useful is `mod_auth_saml`'s  
 1534 ability to "fake" `REMOTE_USER` subprocess environment variable, effectively enabling any application  
 1535 that supports HTTP basic authentication to also support SAML SSO.

1536 We expect to provide specific integration examples for some software packages. As of 2009 none are  
 1537 available, but Mahara is one of the first ones planned.

### 1539 3.3.2 `cf = tas3_new_conf_to_cf(conf)`

1540 **conf** Configuration string

1541 **return** Configuration object

1542

### 1543 3.3.3 `ses = tas3_new_ses(cf)`

1544 Create a new TAS3 session object. Usually a session object is created just before calling

1545 **cf** Configuration object

1546 **return** Session object

1547

### 1548 3.3.4 SSO: `ret = tas3_sso_cf_ses(cf, -1, qs, ses, null, auto_flags)`

1549 **cf** Configuration object, see `tas3_new_conf_to_cf()`

1550 **qs\_len** Length of the query string. -1 = use `strlen()`

1551 **qs** Query string (or POST content)

1552 **ses** Session object, see `tas3_new_ses()`. Session object is modified.

1553 **res\_len** Should always be passed as null (result parameter is not supported for PHP).

1554 **auto\_flags** Automation flags

1555 **return** String representing protocol action or SSO attributes

#### 1556 Example

```
1557 01 <?
1558 02 $qs = $_SERVER['REQUEST_METHOD'] == 'GET'
1559 03     ? $_SERVER['QUERY_STRING']
1560 04     : file_get_contents('php://input');
1561 05 $ses = tas3_new_ses($cf);
```

```

1562 06 $res = tas3_sso_cf_ses($cf, -1, $qs, $ses, null, 0x1814);
1563 07 switch (substr($res, 0, 1)) {
1564 08 case 'L': header($res); exit; # Redirect (Location header)
1565 09 case '<': header('Content-type: text/xml'); echo $res; exit;
1566 10 case 'n': exit; # Already handled
1567 11 case 'e': my_render_idp_select();
1568 12 case 'd': break; # Logged in case
1569 13 default: die("Unknown res($res)");
1570 14 }
1571 15
1572 16 if (tas3_az_cf_ses($cf, "Action=Show", $ses)) {
1573 17     echo "Permit.\n";
1574 18     # Render protected content here
1575 19 } else {
1576 20     echo "<b>Deny.</b>";
1577 21 }
1578 22 ?>
1579

```

### 1580 3.3.5 Authorization: decision = *tas3\_az\_cf\_ses(cf, qs, ses)*

1581 **cf** the configuration object

1582 **qs** additional attributes that are passed to PDP

1583 **ses** session object, from which most attributes come

1584 **return** 0 on deny (for any reason, e.g. indeterminate), or non-null if permit.

1585

### 1586 3.3.6 WSC: resp\_soap = *tas3\_call(cf, ses, svctype, url, di\_opt, az\_cred, req\_soap)*

1587 **cf** Configuration object, see *tas3\_new\_conf\_to\_cf()*

1588 **ses** Session object, used to locate EPRs, see *tas3\_new\_ses()*

1589 **svctype** Service type and namespace URN that is applicable to the body. Passed as a string.

1590 **url** (Optional) If provided, this argument has to match either the ProviderID, EntityID, or actual service  
 1591 endpoint URL.

1592 **di\_opt** (Optional) Additional discovery options for selecting the service, query string format

1593 **az\_cred** (Optional) Additional authorization credentials or attributes, query string format.

1594 **req\_soap** string used as SOAP body or as SOAP envelope template.

1595 **return** SOAP envelope as a string

#### 1596 **Example**

```

1597 01 $ret = tas3_call($cf, $ses, "urn:id-sis-idhrxml:2007-06:dst-2.1",
1598 02     null, null, null,
1599 03     "<idhrxml:Query>" .
1600 04     "    <idhrxml:QueryItem>" .
1601 05     "        <idhrxml:Select>$criteria</idhrxml:Select>" .
1602 06     "    </idhrxml:QueryItem>" .
1603 07     "</idhrxml:Query>");

```

1604

**3.3.7 WSP: tgtid = *tas3\_wsp\_validate(cf, ses, az\_cred, soap\_req)***

 1605 **cf** TAS<sup>3</sup> configuration object, see *tas3\_new\_conf()*  
 1606

 1607 **ses** Session object that contains the EPR cache, see *tas3\_new\_ses()*

 1608 **az\_cred** (Optional) Additional authorization credentials or attributes, query string format.

 1609 **soap\_req** Entire SOAP envelope as a string

 1610 **return** target name id (tgtid), as a string, of the target identity of the request (rest of the information is  
 1611 populated to the session object, from where it can be retrieved).

1612

**3.3.8 WSP: soap = *tas3\_wsp\_decorate(cf, ses, az\_cred, soap\_resp)***

 1613 **cf** TAS<sup>3</sup> configuration object, see *tas3\_new\_conf()*  
 1614

 1615 **ses** Session object that contains the EPR cache

 1616 **az\_cred** (Optional) Additional authorization credentials or attributes, query string format.

 1617 **soap\_resp** XML payload, as a string

 1618 **return** SOAP Envelope of the response, as a string, ready to be sent as HTTP response.

1619

**3.3.9 Explicit Discovery: epr = *tas3\_get\_epr(cf, ses, svc, url, di\_opt, act, n)***

 1620 First search epr cache, and if miss, go discover an EPR over the net. This is the main work horse for  
 1621 WSCs wishing to call WSPs via EPR.  
 1622

 1623 **cf** TAS<sup>3</sup> configuration object, also used for memory allocation

 1624 **ses** Session object in whose EPR cache the file will be searched

 1625 **svc** Service type (usually a URN)

 1626 **url** (Optional) If provided, this argument has to match either the ProviderID, EntityID, or actual service  
 1627 endpoint URL.

 1628 **di\_opt** (Optional) Additional discovery options for selecting the service, query string format

 1629 **act** (Optional) The action, or method, that must be invocable on the service

 1630 **n** Which matching instance is returned. 1 means first

 1631 **return** EPR data structure on success, 0 on failure (no discovery EPR in cache, or not found by the  
 1632 discovery service).  
 1633

**3.3.10 url = *tas3\_get\_epr\_url(cf, epr)***

 1634 **cf** TAS<sup>3</sup> configuration object, also used for memory allocation

 1635 **epr** An EPR object, such as obtained from *tas3\_get\_epr()*

 1636 **return** The <a:Address> field of an EPR as a string. This is the endpoint URL.  
 1637

1638

**1639 3.3.11 entityid = *tas3\_get\_epr\_entid(cf, epr)*****1640 cf** TAS<sup>3</sup> configuration object, also used for memory allocation**1641 epr** An EPR object, such as obtained from *tas3\_get\_epr()***1642 return** The <di:ProviderID> field of an EPR as a string. This is same as SAML2 EntityID.

1643

**1644 3.3.12 a7n = *tas3\_get\_epr\_a7n(cf, epr)*****1645 cf** TAS<sup>3</sup> configuration object, also used for memory allocation**1646 epr** An EPR object, such as obtained from *tas3\_get\_epr()***1647 return** Assertion from EPR <sec:Token> field as a string.

1648

**1649 3.3.13 Available Implementations (Non-normative)****1650** This binding is implemented by `php_zxid` module, available as part of the `zxid.org`

1651

## 1652 3.4 C and C++ Binding

1653 Essentially this is a procedural C binding that is also usable from C++. In fact, the C binding can be  
 1654 used as a base for many other language bindings generated using SWIG [SWIG] interface generator.

1655 The binding is declared in `tas3.h` and implemented in `libtas3.a`, `libtas3.so`, or `libtas3.dll`,  
 1656 depending on the platform. Typical source code file will pull in the TAS<sup>3</sup> API by including

```
1657 #include <tas3.h>
```

1658

### 1659 3.4.1 `cf = tas3_new_conf_to_cf(conf)`

#### 1660 Prototype

```
1661 tas3_conf* tas3_new_conf_to_cf(const char* conf);
```

1662 Create a new TAS3 configuration object given configuration string and possibly configuration file. Usually  
 1663 a configuration object is generated and passed around to different API calls to avoid reparsing the  
 1664 configuration at each API call.

1665 **conf** Configuration string

1666 **return** Configuration object

1667

### 1668 3.4.2 `ses = tas3_new_ses(cf)`

#### 1669 Prototype

```
1670 tas3_ses* tas3_new_conf_to_cf(const char* conf);
```

1671 Create a new TAS3 session object. Usually a session object is created just before calling

1672 **cf** Configuration object

1673 **return** Session object

1674

### 1675 3.4.3 SSO: `ret = tas3_sso_cf_ses(cf, qs_len, qs, ses, &res_len, auto_flags)`

#### 1676 Prototype

```
1677 char* tas3_sso_cf_ses(tas3_conf* cf, int qs_len, char* qs,  
1678                     tas3_ses* ses, int* res_len, int auto_flags);
```

1679 Strings are length + pointer (no C string nul termination needed).

1680 **cf** Configuration object, see `tas3_new_conf_to_cf()`

1681 **qs\_len** Length of the query string. -1 = use `strlen()`

1682 **qs** Query string (or POST content)

1683 **ses** Session object, see `tas3_new_ses()`. Session object is modified.

1684 **res\_len** Result parameter. If non-null, will be set to the length of the returned string

1685 **auto\_flags** Automation flags

1686 **return** String representing protocol action or SSO attributes

1687 **Example**

```

1688 01 {
1689 02 tas3_conf* cf = tas3_new_conf_to_cf("PATH=/var/tas3/");
1690 03 tas3_ses* ses = tas3_new_ses(cf);
1691 04 char* ret = tas3_sso_cf_ses(cf, -1, env("QUERY_STRING"), ses, 0, 0x1800);
1692 05 switch (ret[0]) {
1693 06 case 'd': break; /* Successful login */
1694 07 ... /* Processing other outcomes omitted for brevity. */
1695 08 }
1696 09 if (tas3_az_cf_ses(cf, "", ses)) {
1697 10 /* SSO successful and authorization permit. Do some work. */
1698 11 } else {
1699 12 /* SSO successful but authorization denied */
1700 13 }
1701 14 }
1702
    
```

1703 **3.4.4 Authorization: decision = *tas3\_az\_cf\_ses(cf, qs, ses)***

1704 **Prototype**

```

1705 char* tas3_az_cf_ses(tas3_conf* cf, const char* qs, tas3_ses* ses);
    
```

1706 Call Policy Decision Point (PDP) to obtain an authorization decision about a contemplated action on a  
 1707 resource.

1708 **cf** the configuration object

1709 **qs** additional attributes that are passed to PDP

1710 **ses** session object, from which most attributes come

1711 **return** 0 on deny (for any reason, e.g. indeterminate), or non-null if permit.

1712

1713 **3.4.5 WSC: resp\_soap = *tas3\_call(cf, ses, svctype, url, di\_opt, az\_cred, req\_soap)***

1714 **Prototype**

```

1715 struct zx_str* tas3_call(tas3_conf* cf, tas3_ses* ses, const char* svctype,
1716     const char* url, const char* di_opt, const char* az_cred,
1717     const char* req_soap);
    
```

1718 **cf** Configuration object, see *tas3\_new\_conf\_to\_cf()*

1719 **ses** Session object, used to locate EPRs, see *tas3\_new\_ses()*

1720 **svctype** Service type and namespace URN that is applicable to the body. Passed as a string.

1721 **url** (Optional) If provided, this argument has to match either the ProviderID, EntityID, or actual service  
 1722 endpoint URL.

1723 **di\_opt** (Optional) Additional discovery options for selecting the service, query string format

1724 **az\_cred** (Optional) Additional authorization credentials or attributes, query string format.

1725 **req\_soap** string used as SOAP body or as SOAP envelope template.

1726 **return** SOAP envelope as a string

1727

### 1728 **3.4.6 resp\_soap = tas3\_callf(cf, ses, svctype, url, di\_opt, az\_cred, fmt, ...)**

#### 1729 **Prototype**

```
1730 tas3_str* tas3_callf(tas3_conf* cf, tas3_ses* ses, const char* svctype,
1731     const char* url, const char* di_opt, const char* az_cred,
1732     const char* fmt, ...);
```

1733 The *tas3\_callf()* variant, which allows *printf(3)* style formatting, is highly convenient for C program-  
1734 mers. Others will probably use the plan *tas3\_call()* and rely on language's native abilities to construct the  
1735 string.

1736 **cf** Configuration object, see *tas3\_new\_conf\_to\_cf()*

1737 **ses** Session object, used to locate EPRs, see *tas3\_new\_ses()*

1738 **svctype** Service type and namespace URN that is applicable to the body. Passed as a string.

1739 **url** (Optional) If provided, this argument has to match either the ProviderID, EntityID, or actual service  
1740 endpoint URL.

1741 **di\_opt** (Optional) Additional discovery options for selecting the service, query string format

1742 **az\_cred** (Optional) Additional authorization credentials or attributes, query string format.

1743 **fmt** printf style format string that is used to describe the body of the call as a string. If *fmt* contains format  
1744 specifiers, then additional arguments are used to expand these.

1745 **return** SOAP envelope as a string

1746

### 1747 **3.4.7 WSP: tgtid = tas3\_wsp\_validate(cf, ses, az\_cred, soap\_req)**

#### 1748 **Prototype**

```
1749 char* tas3_wsp_validate(tas3_conf* cf, tas3_ses* ses,
1750     const char* az_cred, const char* soap_req);
```

1751 **cf** TAS<sup>3</sup> configuration object, see *tas3\_new\_conf()*

1752 **ses** Session object that contains the EPR cache, see *tas3\_new\_ses()*

1753 **az\_cred** (Optional) Additional authorization credentials or attributes, query string format.

1754 **soap\_req** Entire SOAP envelope as a string

1755 **return** *idpnid*, as a string, of the target identity of the request (rest of the information is populated to the  
1756 session object, from where it can be retrieved).

1757

### 1758 **3.4.8 WSP: soap = *tas3\_wsp\_decorate*(*cf*, *ses*, *az\_cred*, *soap\_resp*)**

#### 1759 **Prototype**

```
1760 tas3_str* tas3_wsp_decorate(tas3_conf* cf, tas3_ses* ses,
1761                             const char* az_cred, const char* soap_resp);
```

1762 **cf** TAS<sup>3</sup> configuration object, see *tas3\_new\_conf()*

1763 **ses** Session object that contains the EPR cache

1764 **az\_cred** (Optional) Additional authorization credentials or attributes, query string format.

1765 **soap\_resp** XML payload as a string

1766 **return** SOAP Envelope of the response, as a string, ready to be sent as HTTP response.

1767

### 1768 **3.4.9 WSP: soap = *tas3\_wsp\_decoratef*(*cf*, *ses*, *az\_cred*, *fmt*, ...)**

#### 1769 **Prototype**

```
1770 tas3_str* tas3_wsp_decorate(tas3_conf* cf, tas3_ses* ses,
1771                             const char* az_cred, const char* fmt, ...);
```

1772 **cf** TAS<sup>3</sup> configuration object, see *tas3\_new\_conf()*

1773 **ses** Session object that contains the EPR cache

1774 **az\_cred** (Optional) Additional authorization credentials or attributes, query string format.

1775 **fmt** printf style format string that is used to describe the body of the response as a string. If *fmt* contains  
1776 format specifiers, then additional arguments are used to expand these.

1777 **return** SOAP Envelope of the response, as a string, ready to be sent as HTTP response.

1778

### 1779 **3.4.10 Explicit Discovery: epr = *tas3\_get\_epr*(*cf*, *ses*, *svc*, *url*, *di\_opt*, *act*, *n*)**

#### 1780 **Prototype**

```
1781 tas3_epr* tas3_get_epr(tas3_conf* cf, tas3_ses* ses,
1782                        const char* svc, const char* url, const char* di_opt,
1783                        const char* action, int n);
```

1784 First search epr cache, and if miss, go discover an EPR over the net. This is the main work horse for  
1785 WSCs wishing to call WSPs via EPR.

1786 **cf** TAS<sup>3</sup> configuration object, also used for memory allocation

1787 **ses** Session object in whose EPR cache the file will be searched

1788 **svc** Service type (usually a URN)

1789 **url** (Optional) If provided, this argument has to match either the ProviderID, EntityID, or actual service  
1790 endpoint URL.

1791 **di\_opt** (Optional) Additional discovery options for selecting the service, query string format

1792 **act** (Optional) The action, or method, that must be invocable on the service

1793 **n** Which matching instance is returned. 1 means first

1794 **return** EPR data structure on success, 0 on failure (no discovery EPR in cache, or not found by the  
1795 discovery service).

1796

### 1797 **3.4.11 url = *tas3\_get\_epr\_url(cf, epr)***

1798 **Prototype**

1799 `tas3_str* tas3_get_epr_url(tas3_conf* cf, tas3_epr* epr);`

1800 **cf** TAS<sup>3</sup> configuration object, also used for memory allocation

1801 **epr** An EPR object, such as obtained from *tas3\_get\_epr()*

1802 **return** The <a:Address> field of an EPR as a string. This is the endpoint URL.

1803

### 1804 **3.4.12 entityid = *tas3\_get\_epr\_entid(cf, epr)***

1805 **Prototype**

1806 `tas3_str* tas3_get_epr_entid(tas3_conf* cf, tas3_epr* epr);`

1807 **cf** TAS<sup>3</sup> configuration object, also used for memory allocation

1808 **epr** An EPR object, such as obtained from *tas3\_get\_epr()*

1809 **return** The <di:ProviderID> field of an EPR as a string. This is same as SAML2 EntityID.

1810

### 1811 **3.4.13 a7n = *tas3\_get\_epr\_a7n(cf, epr)***

1812 **Prototype**

1813 `tas3_str* tas3_get_epr_a7n(tas3_conf* cf, tas3_epr* epr);`

1814 **cf** TAS<sup>3</sup> configuration object, also used for memory allocation

1815 **epr** An EPR object, such as obtained from *tas3\_get\_epr()*

1816 **return** Assertion from EPR <sec:Token> field as a string.

1817

### 1818 **3.4.14 Available Implementations (Non-normative)**

1819 This binding is implemented, at least, by [zxid.org](http://zxid.org) open source implementation, which serves as the  
1820 reference implementation of the TAS<sup>3</sup> core security architecture.

1821 N.B. The *tas3\_sso()* API is implemented by [zxid](http://zxid.org)'s *zxid\_simple()* API.

1822

### 1823 3.5 Other Language Bindings

1824 At present stage of the TAS<sup>3</sup> project (2009) we only offer Java, PHP, and C/C++ bindings, but in future  
1825 we aim supporting also at least the following

- 1826 • C# / .Net / Mono
- 1827 • Perl (currently `zxid.org` derived `Net::SAML` perl module, available from `cpan.org`, supports most  
1828 functionality of TAS<sup>3</sup> API, but this is unofficial)
- 1829 • Python
- 1830 • Ruby

1831 We welcome external contribution and language specialist help in making all these bindings available.  
1832 Please contact Sampo Kellomäki (`sampo@symlabs.com`) if you are interested.

1833

1834

## 4 Deployment and Integration Models (Non-normative)

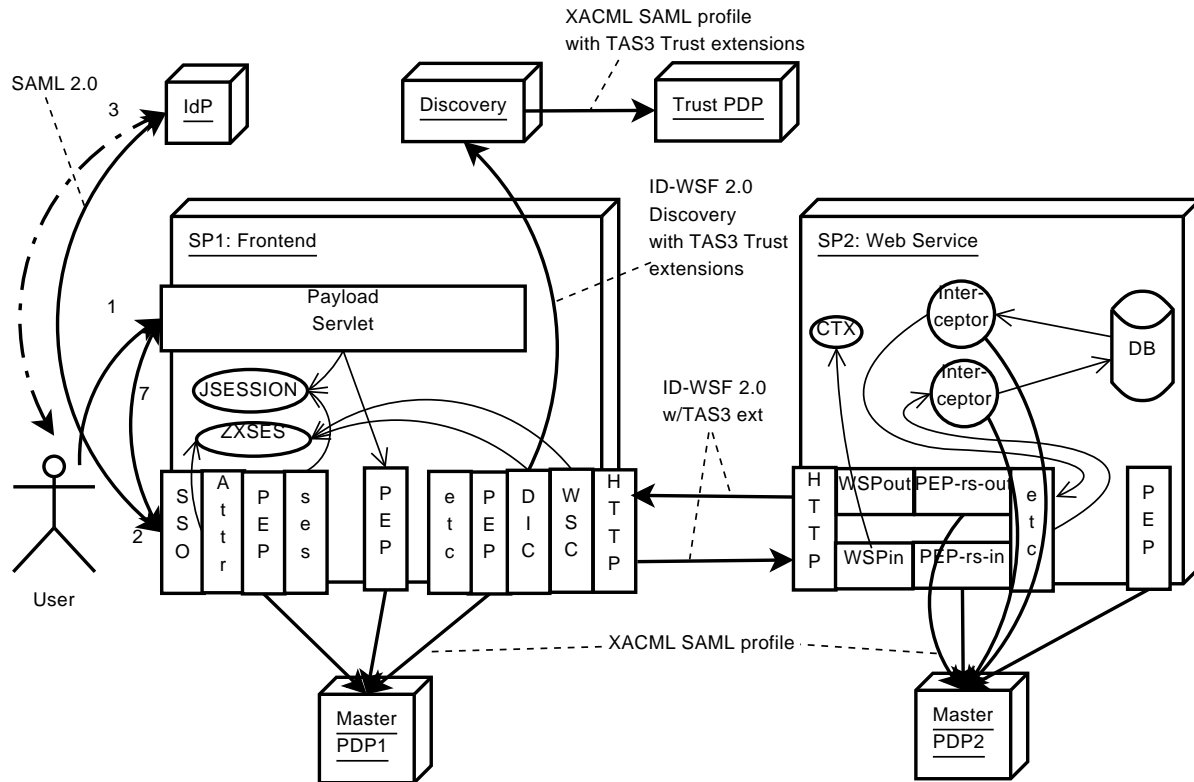


Figure 4.1: A deployment architecture for SSO and web service call.

1835

The above diagram illustrates a typical frontend-backend integration situation.

1836

The TAS<sup>3</sup> integration can be accomplished in several ways, from least intrusive to the original (legacy) application to more intrusive, but also more granular:

1837

1838

**Proxy or mediation box approach** See also [TAS3D71IdMAnAz] Fig-8.2 "Using a Gateway for Legacy Applications". This approach is completely application independent and simply TAS<sup>3</sup> wraps existing protocol. Limitation tends to be that TAS<sup>3</sup> authorization and obligations have to be applied at granularity of a protocol message rather than the data in it.

1839

1840

1841

1842

**Application server filter approach** Either web server module, like mod\_auth\_saml, or an application server module, like Servlet Filter or AXIS2 Interceptor, is inserted to the processing stack. While software realization is quite different, this is still similar to the mediation box model.

1843

1844

1845

**Application class dependent filter approach** Similar to the above filter approach, but the filter has some ability to "drill in" to the application protocol. For example, if all data in the application is represented in uniform format, such as Java Objects, then a generic filter can be supplied that applies authorization and obligations to all data represented in such way.

1846

1847

1848

1849

**API approach** This approach relies the application programmer to instrument his application with necessary authorization and other calls. We are simply trying to make his job easier by providing readily available, TAS<sup>3</sup> certified, APIs that make the instrumenting job easy.

1850

1851

1852

1853

### 4.1 Frontend and Web Services Client Integration Model (Non-normative)

1854

The tasks to be accomplished on the Frontend, in the direct line of call, include

1855

1. Detect need for login (done by payload servlet)

- 1856 2. Perform SSO (SP side)
- 1857 3. Perform SSO, IdP side including authenticating user and shipping attributes
- 1858 4. Gater additional attributes, if needed ("Attr")
- 1859 5. Authorize access to FE (PEP-Rs-In of FE) ("PEP")
- 1860 6. Populate session of the payload servlet ("ses")
- 1861 7. Redirect user to protected resource he was trying to access on the protected resource.
- 1862 8. Application dependent PEP calls PDP if needed. ("PEP")
- 1863 9. Call web service, including
  - 1864 a. Application dependent processing steps ("etc")
  - 1865 b. Authorize the call (PEP-Rq-Out) ("PEP")
  - 1866 c. Discover suitable service, performing Trust and Privacy Negotiation (may need interaction at front-end web gui) if needed. ("DIC")
  - 1867 d. Decorate request with TAS3 specific SOAP headers and sign. ("WSC")
- 1869 10. Perform network I/O ("HTTP"). This also includes TLS certificate authentication of the Responder and may include Client-TLS certificate authentication of the Requester.

1871 The SSO integration is expected to be a single module, appearing as a servlet in Java realization and as an authentication module in web server realization, that handles steps 2-7 automatically. The integration is accomplished by configuring the web server without modifying the application except to add the initial detection and redirect (1) and to make use of the attributes that were populated to the session.<sup>1</sup> The TAS<sup>3</sup> binary modules for SSO are generically called T3-SSO-\*

1876 The WSC integration is expected to be a single module. It will appear as AXIS2 module in Java realization so that it can be just hooked in by configuration without any modification to the existing web service (the "etc" module illustrates that even other modules than TAS<sup>3</sup> can be hooked in without interference<sup>2</sup>).

1879 The API realization of WSC is a function, *tas3\_call()* (see TAS<sup>3</sup> API), that the application can call directly. If this approach is chosen, the entire web services call is handled by the API without any regard to servlet environment's or framework's hooking or modules. This is the most common approach in PHP, Perl, C#, C++, and C worlds.

1883 A possible variant of WSC integration is to call *tas3\_call\_prepare()* to obtain the serialized SOAP envelope, then do the I/O part in application dependent way, and pass the response to *tas3\_response\_validate()*. Effectively *tas3\_call()* does these steps with a built-in HTTP client performing the I/O part.<sup>3</sup>

#### 1887 4.1.1 Integration Using ZXID (Non-normative)

1888 Further information about using ZXID for TAS<sup>3</sup> is available in README.zxid-tas3, zxid-tas3.pd, and zxid-java.pd

1890 The official TAS<sup>3</sup> API is provided by *tas3.h* which maps the TAS<sup>3</sup> API definitions to the underlying zxid ones.

1892 The Java realization of SSO is provided by *zxidservlet* class and *servlet*. This is packaged as TAS<sup>3</sup> binary module T3-SSO-ZXID-JAVA.

1894 The web server realization of SSO is provided by *mod\_auth\_saml* Apache module (*mod\_auth\_saml.so*). It is packaged as TAS<sup>3</sup> binary module T3-SSO-ZXID-MODAUTHSAML.

<sup>1</sup>In *mod\_auth\_saml* realization even step (1) can be accomplished by configuring the web server.

<sup>2</sup>Non-interference depends on other modules following certain common sense conventions, such as not signing SOAP <e:Headers> element and not trying to create SOAP headers that TAS3 creates (e.g. <wsse:Security>).

<sup>3</sup>In ZXID realization the HTTP client is libcurl from curl.haxx.se

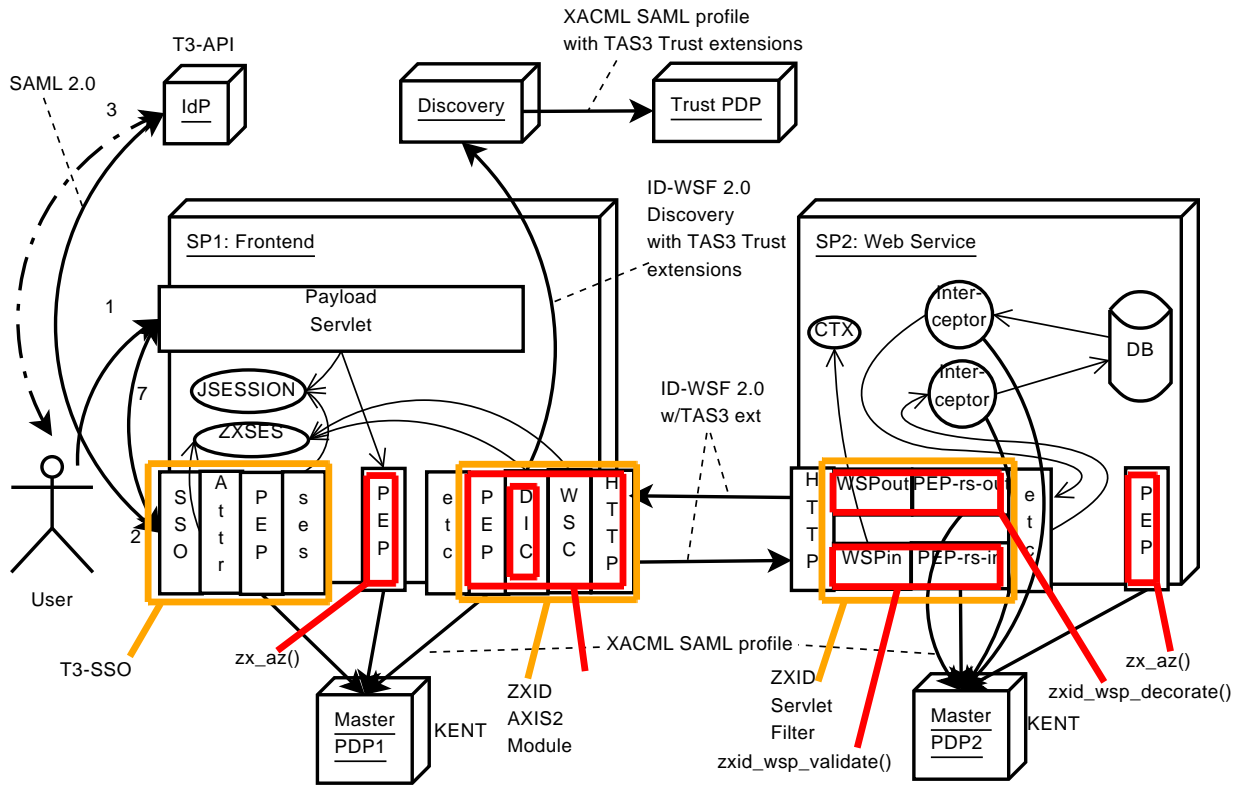


Figure 4.2: API and modules for SSO and web service call.

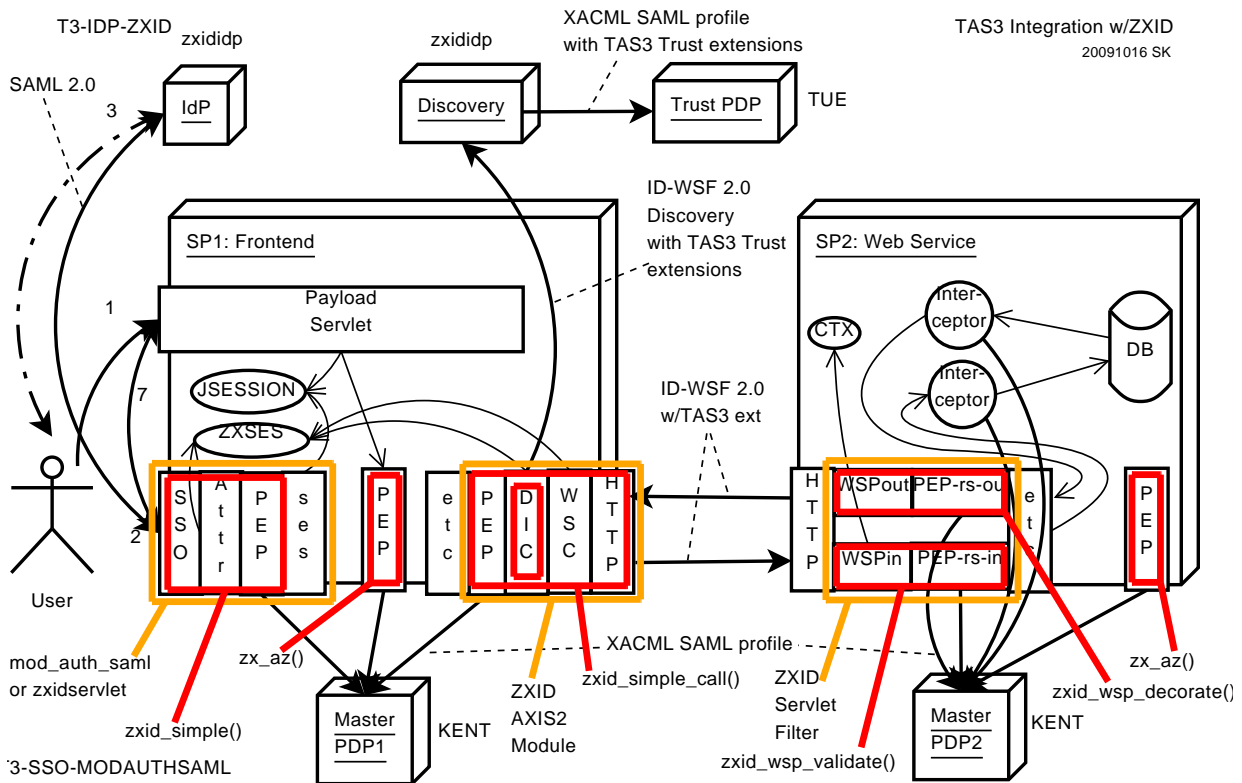


Figure 4.3: ZXID specific API and modules for SSO and web service call.

1896 API realization of SSO is provided by *zxid\_simple()* in *libzxid.a*. This is packaged as TAS<sup>3</sup> binary  
 1897 module T3-SSO-ZXID-PHP.<sup>4</sup> Other language binding specific modules are expected in the future.

<sup>4</sup>Although not TAS3 packaged, Net::SAML perl module provides the same functionality.

1898

1899 **4.1.2 Integration Using Other Platforms, Frameworks, and Packages (Non-**  
 1900 **normative)**

1901 Other mainstream packages are invited to submit integration descriptions similar to previous section  
 1902 (ZXID). The details of the integration should be in package's own documentation.

1904 **4.2 Web Services Provider Integration Model (Non-normative)**

1905 The tasks to be accomplished on the Service Responder, in the direct line of call, include

- 1906 A. Listen for HTTP requests (typically done by platform)
- 1907 B. Parse and validate a web services request, e.g. call *tas3\_wsp\_validate()*. This involves checking for  
 1908 valid signature from trusted authority.
- 1909 C. Authorize the request, extracting from the request the pledges (in `<b:UsageDirective>`) ("PEP-Rs-  
 1910 In").
- 1911 D. Apply other filters and post processing steps ("etc")
- 1912 E. Authorize each data item separately using input interceptor. For queries this is usually a no-op, but for  
 1913 creates or updates this is meaningful. When data is accepted for the repository, the authorization step  
 1914 can result in obligations or sticky-policies being written into the database along side the data itself.  
 1915 The authorization is configurable according to Application Independent PEP configuration, described  
 1916 elsewhere, or Application Dependent PEP approach can be taken, calling the PDP directly ("PEP").
- 1917 F. Authorize each returned data item separately using input interceptor. Usually applicable to query  
 1918 results. The per item authorization will apply systemwide and item specific policies (sticky policies)  
 1919 and obligations and produce a deny or permit-with-obligations response.  
 1920 The authorization is configurable according to Application Independent PEP configuration, described  
 1921 elsewhere, or Application Dependent PEP approach can be taken, calling the PDP directly ("PEP").
- 1922 G. Authorize the response in aggregate ("PEP-Rs-Out"). At this stage one of the most important veri-  
 1923 fications is to compare the pledges collected in step C ("PEP-Rs-In") and filter out any data whose  
 1924 obligations are stricter.

1925 **Optimization.** It is possible to combine the pledges to obligations matching (in G) to the  
 1926 per result item authorization (F) by simply feeding the pledges as inputs to the PDP in (F).  
 1927 Such optimization can not, however, achieve all functionality of the G ("PEP-Rs-Out") as it  
 1928 is unable to see the bigger picture, i.e. consider all data together as a set. A typical example  
 1929 would be a rule against leaking simultaneously day and month of birth and year of birth.

- 1930 H. Decorate the response with TAS<sup>3</sup> specific SOAP headers. This is typically done by calling *tas3\_wsp\_decorate()*.
- 1931 I. Send the response. This is typically done by platform dependent means.

1932

1933

## 5 Resilient Deployment Architecture (Non-normative)

1934

This section addresses Req. *DI.2-2.8-Avail.*

1935

1936

1937

1938

1939

For TAS<sup>3</sup> services to be dependable, they need to be deployed so that they are resilient to system and network failure. Resiliency and efficiency are the first lines of defense against Denial of Service attacks that try to attack simple catastrophic vulnerabilities or overwhelm the system on the point where it is most inefficient. Resiliency needs to be considered at several layers, namely on the Front Channel and on the Back Channel.

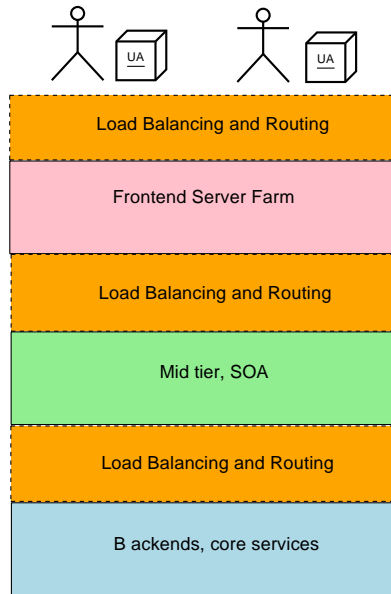


Figure 5.1: Layering of resilience features for Front Channel, Back Channel, and data center Back End services.

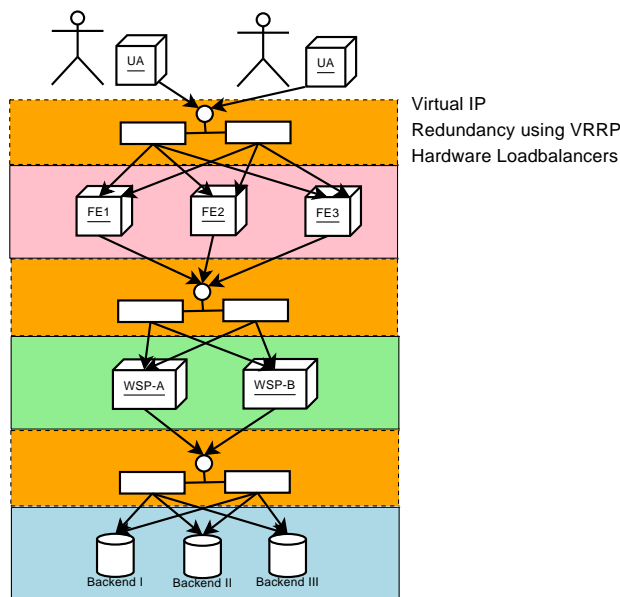


Figure 5.2: Resiliency implemented using hardware load balancers.

1940

1941

Note that the virtual IP address is hosted either in hardware load balancer, or one member of a cluster. Fail-over of the virtual IP is arranged using Virtual Router Redundancy Protocol (VRRP) [RFC3768].

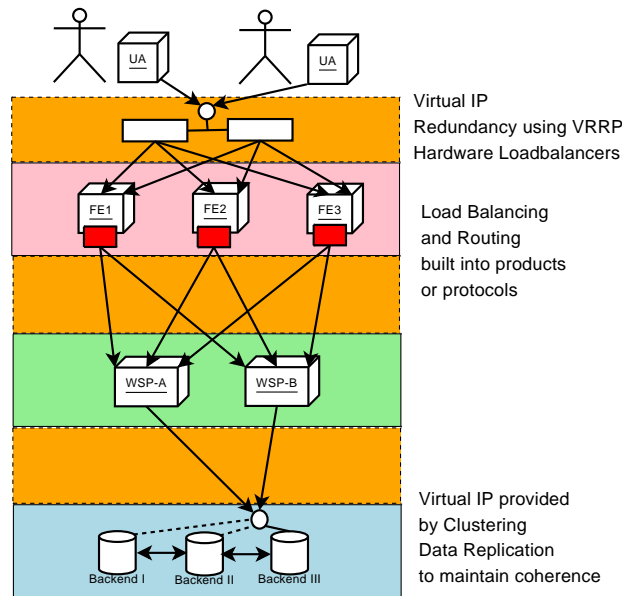


Figure 5.3: Resiliency implemented using software load-balancing-fail-over functionality and clustering.

1942

## 1943 5.1 Zero Downtime Updates

1944 This section addresses Req. *D1.2-7.19-DynaUpd*.

1945 For continued availability of the system, Zero-Downtime-Update (ZDTU) technology SHOULD be  
 1946 implemented through out. If horizontal scaling path and failure recovery have been implemented, then  
 1947 ZDTU can be implemented easily by taking out of farm one server at a time and updating it. Downside of  
 1948 this approach is that the farm will temporarily be in an inconsistent state.

1949 If consistency of the farm is at all times a requirement, no easy ZDTU approach exists. One approach  
 1950 is to bring up new "hot standbys" along side of the old configuration and then do instantaneous switch. As  
 1951 the switch over is less than 1 second, this could be considered ZDTU.

1952 Never-the-less, as TAS<sup>3</sup> is business process driven and as business processes can take long time to  
 1953 complete (if human interaction is required, this could easily mean days or weeks), thus consistent ZDTU  
 1954 is infeasible in practise and the business process modelling should explicitly foresee handling of upgrade  
 1955 situations, i.e. how old processes are handled after the general upgrade.

1956

1957

## 6 Feasibility and Performance Analysis (Non-normative)

1958

1959

TAS<sup>3</sup> Architecture is rather complex so we need to analyze the runtime cost of implementing it. The cost can be divided in six categories

1960

1961

1962

1963

1964

1965

1966

1967

1968

1969

1970

**T** Connection overhead, including TCP handshake and TLS handshake. The latter involves one public key operation on both sides, unless TLS connection cache hit is achieved. Except for the cache hit case, connection overhead is mostly unavoidable given TAS<sup>3</sup> Architecture's division of components. Sometimes co-locating several components in same host may allow use of localhost connection to avoid handshake overhead. The TLS overhead may be avoidable in localhost and secure internal network cases. The TCP overhead is very sensitive to latency: usually a precondition for a connection is to resolve a domain name: this means one round trip latency cost. Then actual threeway TCP handshake needs to be performed, causing three round trip latencies. Finally TLS handshake causes at least one more round trip. Therefore the time cost of a connection tends to be minimum of 5 round trip latencies. Higher the latency, more time it takes to process a call and more simultaneous calls are needed to keep up the same through put.

1971

1972

1973

1974

**C** Communication overhead: this consists of compression, encryption (symmetric stream cipher), and transfer of the actual data. Mostly unavoidable. As communication cost and stream cipher tend to be negligible compared to TCP + TLS handshake and digital signatures, we will not consider communication cost in our calculations.

1975

1976

1977

1978

**S** Digital signature overhead: usually at least one public key operation is involved on each side. Often responder side needs to verify several digital signatures: one for the message and one for each token or credential it receives. The signature overhead is mostly unavoidable, though some caching and session techniques may reduce it in case of often repeated actions.

1979

1980

1981

1982

1983

1984

**X** XML overhead: the arcane and poorly designed features, such as namespaces and canonicalization, of XML cause significant processing overhead (not to mention bugs). In some Java implementations of digital signature processing the XML formatting consumes as much CPU as the public key operation. Even in the best of breed implementations XML formatting has significant cost, usually about 20% of the cost of a public key operation. XML cost could be eliminated by choosing a more rational data format.

1985

1986

1987

1988

**Z** Authorization cost. Evaluation of rule set will depend heavily on the particular ruleset and its implementation technology. Some rulesets are known to take exponential time to evaluate. Authorization cost is exclusively borne by the PDP components. While a PDP may incur additional cost in validating credentials, this is not taken in account here (but can be accounted as digital signature overhead).

1989

1990

**P** Payload cost. This is the cost of running the actual application and is unavoidable. Since we are trying to measure the overhead cost of TAS<sup>3</sup> Architecture, the payload is assumed to be free.

1991

In cost calculations we will use units with overall cost computed as show in following table:

1992

1993

1994

1995

The cost is unevenly divided among the entities in the TAS<sup>3</sup> trust network, but the division depends heavily on whether caching can be utilized. If the usage pattern is isolated single operations, the IdP, discovery, and credential issuance tend to become hotspots because these functions are relied on by many other players in the network. For single operations the TLS cache misses will penalize the system overall.

1996

1997

1998

If the usage pattern is repeat operations, then the bottleneck tends to shift towards responder processing: credentials can be cached, but they still need to be validated every time (some checksum based validation cache may be feasible, but has not been explored yet).

1999

2000

2001

Overall bottlenecks in both cases include audit bus logging, local audit trail (especially if digitally signed), and authorization. In this analysis audit bus is assumed to work by exchanging digitally signed SOAP messages and each exchange to be authorized separately.

2002

To explore the cost we will consider two scenarios.

Table 6.1: Units of cost computation and their RSA equivalence

Unit	RSA Eq.	Definition
T	1.5	One TLS connection establishment. Not entirely RSA comparable as latency component is involved.
t	0.5	One TLS connection establishment, with connection cache hit (avoids public key operation)
S	1	One digital signature generation or validation
X	1	One XML document parse or canonicalization
Z	0.5	One ruleset evaluation.

2003

2004

## 6.1 Single use of single web service

2005

This scenario consists of user making Single Sign-On to a frontend and invoking an operation that requires calling a web service. The sequence of events and the cost is indicated in the table.

2006

2007

Table 6.1: Cost of TAS<sup>3</sup> single use scenario

Operation	IdP + Disc.	Frontend	FE PDP	Responder	Rs PDP	Audit Bus	Audit Bus PDP
1. SSO	2T+4S+4X=11	4T+3S+5X=14	2T+2S+3X+Z=8.5			4(2T+S+3X)=28	4(T+2X+Z)=16
2. Discovery	2T+3S+3X=9	T+S+X=3.5				2T+S+3X=7	t+2X+Z=2.5
3. Trust & Priv.	T+2X=3.5				2T+S+3X=7	2T+S+3X=7	t+2X+Z=2.5
4. Rq Out PEP		t+2X=2	2t+2S+4X+1Z=6.5			2t+S+3X=4	t+2X+Z=2.5
5. Send request		2T+2S+2X=7		2T+3S+3X=9		2(2t+S+3X)=8	2(t+2X+Z)=5
6. Rs In PEP				t+2X=2	2t+2S+4X+Z=6.5	2t+S+3X=4	t+2X+Z=2.5
7. Payload							
8. Rs Out PEP				t+2X=2	2t+2S+4X+Z=6.5	2t+S+3X=4	t+2X+Z=2.5
9. Send response		t+2S+2X=4		t+2S+2X=4		2(2t+S+3X)=8	2(t+2X+Z)=5
10. Rq In PEP		t+2X=2	2t+2S+4X+Z=6.5			2t+S+3X=4	t+2X+Z=2.5
11 Process Oblig		2t+S+2X=3		2t+S+2X=3		2(2t+S+3X)=8	2(t+2X+Z)=5
12. SLO	2t+2S+3X=5	2t+2S+3X=5				2(2t+S+3X)=8	2(t+2X+Z)=5
TOTAL	5T+9S+12X=28	5T+11S+19X=40	52T+6S+11X+3Z=21	5T+6S+11X=20	2T+5S+11X+2Z=20	12T+18S+54X=90	4T+36X+18Z=51

2008

2009

The grand total is  $34T+55S+154X+23Z=271.5$  RSA operation equivalents.

2010

For a fair comparison, a simple web service call without any authorization or auditing, using HTTP Basic authentication and TLS, the cost is shown in the following table. The total cost of such unsecure call is estimated as 8.5 RSA operation equivalents. The cost of a fully secure platform appears to be about 31 times that of unsecure platform.

2011

2012

2013

Table 6.2: Cost of unsecure single use scenario

Operation	Frontend	Responder
1. Login	T=1.5	
5. Send request	T+X=2.5	T+X=2.5
7. Payload		0
9. Send response	X=1	X=1
TOTAL	2T+S+2X=5	1T+S+2X=3.5

2014

2015

2016

2017 **6.1.1 Cost without auditing**

2018 Above calculation shows that the Audit Bus substantially adds to the cost. Here's the same calculation  
2019 without Audit Bus.

2020 Table 6.3: Cost of TAS<sup>3</sup> single use scenario without auditing

Operation	IdP + Disc.	Frontend	FE PDP	Responder	Rs PDP
1. SSO	$1T+2S+2X=5.5$	$3T+2S+4X=10.5$	$T+S+2X+Z=5$		
2. Discovery	$1T+2S+2X=5.5$	$T+S+X=3.5$			
3. Trust & Priv.	$T+2X=3.5$				$T+2X=3.5$
4. Rq Out PEP		$T+2X=3.5$	$1T+1S+3X+1Z=6$		
5. Send request		$1T+1S+1X=3.5$		$1T+2S+1X=4.5$	
6. Rs In PEP				$T+2X=3.5$	$1T+1S+3X+1Z=6$
7. Payload				0	
8. Rs Out PEP				$T+2X=3.5$	$1T+1S+3X+1Z=6$
9. Send response		$S+X=2$		$S+X=2$	
10. Rq In PEP		$T+2X=3.5$	$T+S+3X+Z=6$		
11. Process Obli		$T+X=2.5$		$T+X=2.5$	
12. SLO	$T+S+2X=4.5$	$T+S+2X=4.5$			
TOTAL	$4T+5S+8X=19$	$9T+6S+14X=33.5$	$3T+3S+8X+3Z=17$	$4T+3S+7X=16$	$3T+2S+8X+2Z=15.5$

2021

2022 The grand total without auditing is  $23T+19S+45X+5Z=101$  RSA operation equivalents. As can be  
2023 seen, the Audit Bus represents 63% of the total cost. Most of the Audit Bus cost is actually caused by  
2024 requirement to contact the bus and authorize the sending of messages. A future revision of the architecture  
2025 will explore the possibility of persistent connection to the Audit Bus. This would significantly reduce the  
2026 T, t, S, and Z aspects of the Audit Bus processing, though at least one signature overhead will be needed  
2027 at the message source to ensure untamperability of the audit trail.

2028 Another optimization would be to improve the authorization step of the Audit Bus, perhaps co-locating  
2029 the Audit Bus PDP with the Audit Bus itself.

2031 **6.1.2 Cost without auditing and without authorization**

2032 Another recurring activity are the frequent calls to the PDPs. Following table explores how much could  
2033 be saved by optimising these calls.

2034 Table 6.4: Cost of TAS<sup>3</sup> single use scenario without auditing and without authorization

Operation	IdP + Disc.	Frontend	Responder
1. SSO	$1T+2S+2X=5.5$	$3T+2S+4X=10.5$	
2. Discovery	$1T+2S+2X=5.5$	$T+S+X=3.5$	
5. Send request		$1T+1S+1X=3.5$	$1T+2S+1X=4.5$
7. Payload			
9. Send response		$S+X=2$	$S+X=2$
11. Process Oblig		$T+X=2.5$	$T+X=2.5$
12. SLO	$T+S+2X=4.5$	$T+S+2X=4.5$	
TOTAL	$3T+5S+6X=15.5$	$7T+6S+10X=26.5$	$2T+3S+3X=9$

2035

2036 The grand total without audit and without authorization is  $12T+14S+19X+0Z=51$  RSA operation equiv-  
2037 alents. The authorization steps (excluding Audit Bus related authorization) seem to be adding about as  
2038 much over head as the entire rest of the web service call.

2039 The bare ID-WSF 2.0 web service call compares relatively favorably with bare unsecure web service  
2040 call: 51 vs. 8.5 - only 6 times heavier.

2041

2042 **6.1.3 Cost without XML**

2043 Since XML processing is needlessly expensive, lets analyze what the cost could be with non-XML  
 2044 protocols like RESTful approach using Simple Web Tokens [Hardt09].

2045 Table 6.5: Cost of TAS<sup>3</sup> single use scenario without XML

Operation	IdP + Disc	Frontend	FE PDP	Responder	Rs PDP	Audit Bus	Audit Bus PDP
1. SSO	2T+4S=7	4T+3S=9	2T+2S+Z=5.5			4(2T+S)=16	4(T+Z)=8
2. Discovery	2T+3S=6	T+S=2.5				2T+S=4	T+Z=2
3. Trust & Priv.	T=1.5				2T+S=4	2T+S=4	T+Z=2
4. Rq Out PEP		T=1.5	2T+2S+Z=5.5			2T+S=4	T+Z=2
5. Send request		2T+2S=5		2T+3S=6		2(2T+S)=8	2(T+Z)=4
6. Rs In PEP				T=1.5	2T+2S+Z=5.5	2T+S=4	T+Z=2
7. Payload							
8. Rs Out PEP				T=1.5	2T+2S+Z=5.5	2T+S=4	T+Z=2
9. Send response		T+2S=3.5		T+2S=3.5		2(2T+S)=8	2(T+Z)=4
10. Rq In PEP		T=1.5	2T+2S+Z=5.5			2T+S=4	T+Z=2
11. Process Obli		2T+S=4		2T+S=4		2(2T+S)=8	2(T+Z)=4
12. SLO	2T+2S=5	2T+2S=5				2(2T+S)=8	2(T+Z)=4
2046 TOTAL	7T+9S=19.5	14T+11S=32	6T+6S+3Z=16.5	7T+6S=16.5	6T+5S+2Z=15	36T+18S=72	18T+S+X+18Z=36

2047 Without the XML, but otherwise fully featureful architecture leads to grand total of 94T+55S+0X+23Z=207.5  
 2048 RSA equivalents. Thus eliminating XML can lead to over 40% of savings.

2050 **6.2 Session of 3 frontends and five web services**

2051 This session is meant to illustrate the types of savings available from caching discovery results.

2052 The three frontends are all accessed in the same single sign-on session, leading to savings at IdP. Each  
 2053 frontend then calls two web services. One (A) is common, shared web service. Other (B) is new web  
 2054 service (new for each frontend), but the service is called 4 times, which leads to EPR cache hits. The  
 2055 pattern also encourages TLS cache hits. We also assume repeated calls to PDP and audit bus lead to TLS  
 2056 cache hits.

2057 Table 6.6: Cost of TAS<sup>3</sup> multi use scenario

Operation	IdP + Disc.	Frontend	FE PDP	Responders	Rs PDPs	Audit Bus	Audit Bus PDP
1. SSO w/auth	2T+4S+4X=11	4T+3S+5X=14	2T+2S+3X+Z=8.5			4(2T+S+3X)=28	4(t+2X+Z)=10
2. Discovery A	2t+3S+3X=6	T+S+X=3.5				2t+S+3X=4	t+2X+Z=2.5
3. Trust & Priv.	T+2X=3.5				2T+S+3X=7	2T+S+3X=7	t+2X+Z=2.5
4. Rq Out PEP		t+2X=2	2t+2S+4X+1Z=6.5			2t+S+3X=4	t+2X+Z=2.5
5. Send request		T+t+2S+2X=5.5		T+t+3S+3X=7.5		2(2t+S+3X)=8	2(t+2X+Z)=5
6. Rs In PEP				T+2X=3.5	2T+2S+4X+Z=9.5	2t+S+3X=4	t+2X+Z=2.5
7. Payload							
8. Rs Out PEP				t+2X=2	2t+2S+4X+Z=6.5	2t+S+3X=4	t+2X+Z=2.5
9. Send response		t+2S+2X=4		t+2S+2X=4		2(2t+S+3X)=8	2(t+2X+Z)=5
10. Rq In PEP		t+2X=2	2t+2S+4X+Z=6.5			2t+S+3X=4	t+2X+Z=2.5
11. Process Obli		2t+S+2X=3		2t+S+2X=3		2(2t+S+3X)=8	2(t+2X+Z)=5
12. Discovery B	2t+3S+3X=6	T+S+X=3.5				2t+S+3X=4	t+2X+Z=2.5
13. Trust & Priv.	T+2X=3.5				2T+S+3X=7	2T+S+3X=7	t+2X+Z=2.5
14. Rq Out PEP		t+2X=2	2t+2S+4X+1Z=6.5			2t+S+3X=4	t+2X+Z=2.5
15. Send request		T+t+2S+2X=5.5		T+t+3S+3X=7.5		2(2t+S+3X)=8	2(t+2X+Z)=5
16. Rs In PEP				T+2X=3.5	2T+2S+4X+Z=9.5	2t+S+3X=4	t+2X+Z=2.5
17. Payload							
18. Rs Out PEP				t+2X=2	2t+2S+4X+Z=6.5	2t+S+3X=4	t+2X+Z=2.5
19. Send respons		t+2S+2X=4		t+2S+2X=4		2(2t+S+3X)=8	2(t+2X+Z)=5
20. Rq In PEP		t+2X=2	2t+2S+4X+Z=6.5			2t+S+3X=4	t+2X+Z=2.5
21. Process Obli		2t+S+2X=3		2t+S+2X=3		2(2t+S+3X)=8	2(t+2X+Z)=5
22. Rq Out PEP		t+2X=2	2t+2S+4X+1Z=6.5			2t+S+3X=4	t+2X+Z=2.5
23. Send request		2t+2S+2X=4		2t+3S+3X=6		2(2t+S+3X)=8	2(t+2X+Z)=5
24. Rs In PEP				t+2X=2	2t+2S+4X+Z=6.5	2t+S+3X=4	t+2X+Z=2.5
25. Payload							
26. Rs Out PEP				t+2X=2	2t+2S+4X+Z=6.5	2t+S+3X=4	t+2X+Z=2.5
27. Send respons		t+2S+2X=4		t+2S+2X=4		2(2t+S+3X)=8	2(t+2X+Z)=5
28. Rq In PEP		t+2X=2	2t+2S+4X+Z=6.5			2t+S+3X=4	t+2X+Z=2.5
2058 29. Process Obli		2t+S+2X=3		2t+S+2X=3		2(2t+S+3X)=8	2(t+2X+Z)=5

2059

Table 6.6 (continued): Cost of TAS<sup>3</sup> multi use scenario

Operation	IdP + Disc.	Frontend	FE PDP	Responders	Rs PDPs	Audit Bus	Audit Bus PDP
30. Rq Out PEP		t+2X=2	2t+2S+4X+1Z=6.5			2t+S+3X=4	t+2X+Z=2.5
31. Send request		2t+2S+2X=4		2t+3S+3X=6		2(2t+S+3X)=8	2(t+2X+Z)=5
32. Rs In PEP				t+2X=2	2t+2S+4X+Z=6.5	2t+S+3X=4	t+2X+Z=2.5
33. Payload							
34. Rs Out PEP				t+2X=2	2t+2S+4X+Z=6.5	2t+S+3X=4	t+2X+Z=2.5
35. Send respons		t+2S+2X=4		t+2S+2X=4		2(2t+S+3X)=8	2(t+2X+Z)=5
36. Rq In PEP		t+2X=2	2t+2S+4X+Z=6.5			2t+S+3X=4	t+2X+Z=2.5
37. Process Obli		2t+S+2X=3		2t+S+2X=3		2(2t+S+3X)=8	2(t+2X+Z)=5
38. Rq Out PEP		t+2X=2	2t+2S+4X+1Z=6.5			2t+S+3X=4	t+2X+Z=2.5
39. Send request		2t+2S+2X=4		2t+3S+3X=6		2(2t+S+3X)=8	2(t+2X+Z)=5
40. Rs In PEP				t+2X=2	2t+2S+4X+Z=6.5	2t+S+3X=4	t+2X+Z=2.5
41. Payload							
42. Rs Out PEP				t+2X=2	2t+2S+4X+Z=6.5	2t+S+3X=4	t+2X+Z=2.5
43. Send respons		t+2S+2X=4		t+2S+2X=4		2(2t+S+3X)=8	2(t+2X+Z)=5
44. Rq In PEP		t+2X=2	2t+2S+4X+Z=6.5			2t+S+3X=4	t+2X+Z=2.5
45. Process Obli		2t+S+2X=3		2t+S+2X=3		2(2t+S+3X)=8	2(t+2X+Z)=5
46. SSO ses act	t+4S+4X=8	4T+3S+5X=14	2T+2S+3X+Z=8.5			4(2T+S+3X)=28	4(t+2X+Z)=10
47. Discovery A	2t+3S+3X=6	T+S+X=3.5				2t+S+3X=4	t+2X+Z=2.5
48. Trust & Priv.	T+2X=3.5				2T+S+3X=7	2T+S+3X=7	t+2X+Z=2.5
49. Rq Out PEP		t+2X=2	2t+2S+4X+1Z=6.5			2t+S+3X=4	t+2X+Z=2.5
50. Send request		T+t+2S+2X=5.5		T+t+3S+3X=7.5		2(2t+S+3X)=8	2(t+2X+Z)=5
51. Rs In PEP				T+2X=3.5	2T+2S+4X+Z=9.5	2t+S+3X=4	t+2X+Z=2.5
52. Payload							
53. Rs Out PEP				t+2X=2	2t+2S+4X+Z=6.5	2t+S+3X=4	t+2X+Z=2.5
54. Send respons		t+2S+2X=4		t+2S+2X=4		2(2t+S+3X)=8	2(t+2X+Z)=5
55. Rq In PEP		t+2X=2	2t+2S+4X+Z=6.5			2t+S+3X=4	t+2X+Z=2.5
56. Process Obli		2t+S+2X=3		2t+S+2X=3		2(2t+S+3X)=8	2(t+2X+Z)=5
57. Discovery C	2t+3S+3X=6	T+S+X=3.5				2t+S+3X=4	t+2X+Z=2.5
58. Trust & Priv.	T+2X=3.5				2T+S+3X=7	2T+S+3X=7	t+2X+Z=2.5
59. Rq Out PEP		t+2X=2	2t+2S+4X+1Z=6.5			2t+S+3X=4	t+2X+Z=2.5
60. Send request		T+t+2S+2X=5.5		T+t+3S+3X=7.5		2(2t+S+3X)=8	2(t+2X+Z)=5
61. Rs In PEP				T+2X=3.5	2T+2S+4X+Z=9.5	2t+S+3X=4	t+2X+Z=2.5
62. Payload							
63. Rs Out PEP				t+2X=2	2t+2S+4X+Z=6.5	2t+S+3X=4	t+2X+Z=2.5
64. Send respons		t+2S+2X=4		t+2S+2X=4		2(2t+S+3X)=8	2(t+2X+Z)=5
65. Rq In PEP		t+2X=2	2t+2S+4X+Z=6.5			2t+S+3X=4	t+2X+Z=2.5
66. Process Obli		2t+S+2X=3		2t+S+2X=3		2(2t+S+3X)=8	2(t+2X+Z)=5
67. Rq Out PEP		t+2X=2	2t+2S+4X+1Z=6.5			2t+S+3X=4	t+2X+Z=2.5
68. Send request		2t+2S+2X=4		2t+3S+3X=6		2(2t+S+3X)=8	2(t+2X+Z)=5
69. Rs In PEP				t+2X=2	2t+2S+4X+Z=6.5	2t+S+3X=4	t+2X+Z=2.5
70. Payload							
71. Rs Out PEP				t+2X=2	2t+2S+4X+Z=6.5	2t+S+3X=4	t+2X+Z=2.5
72. Send respons		t+2S+2X=4		t+2S+2X=4		2(2t+S+3X)=8	2(t+2X+Z)=5
73. Rq In PEP		t+2X=2	2t+2S+4X+Z=6.5			2t+S+3X=4	t+2X+Z=2.5
74. Process Obli		2t+S+2X=3		2t+S+2X=3		2(2t+S+3X)=8	2(t+2X+Z)=5
75. Rq Out PEP		t+2X=2	2t+2S+4X+1Z=6.5			2t+S+3X=4	t+2X+Z=2.5
76. Send request		2t+2S+2X=4		2t+3S+3X=6		2(2t+S+3X)=8	2(t+2X+Z)=5
77. Rs In PEP				t+2X=2	2t+2S+4X+Z=6.5	2t+S+3X=4	t+2X+Z=2.5
78. Payload							
79. Rs Out PEP				t+2X=2	2t+2S+4X+Z=6.5	2t+S+3X=4	t+2X+Z=2.5
80. Send respons		t+2S+2X=4		t+2S+2X=4		2(2t+S+3X)=8	2(t+2X+Z)=5
81. Rq In PEP		t+2X=2	2t+2S+4X+Z=6.5			2t+S+3X=4	t+2X+Z=2.5
82. Process Obli		2t+S+2X=3		2t+S+2X=3		2(2t+S+3X)=8	2(t+2X+Z)=5
83. Rq Out PEP		t+2X=2	2t+2S+4X+1Z=6.5			2t+S+3X=4	t+2X+Z=2.5
84. Send request		2t+2S+2X=4		2t+3S+3X=6		2(2t+S+3X)=8	2(t+2X+Z)=5
85. Rs In PEP				t+2X=2	2t+2S+4X+Z=6.5	2t+S+3X=4	t+2X+Z=2.5
86. Payload							
87. Rs Out PEP				t+2X=2	2t+2S+4X+Z=6.5	2t+S+3X=4	t+2X+Z=2.5
88. Send respons		t+2S+2X=4		t+2S+2X=4		2(2t+S+3X)=8	2(t+2X+Z)=5
89. Rq In PEP		t+2X=2	2t+2S+4X+Z=6.5			2t+S+3X=4	t+2X+Z=2.5
90. Process Obli		2t+S+2X=3		2t+S+2X=3		2(2t+S+3X)=8	2(t+2X+Z)=5

2060

2061

Table 6.6 (continued): Cost of TAS<sup>3</sup> multi use scenario

Operation	IdP + Disc.	Frontend	FE PDP	Responders	Rs PDPs	Audit Bus	Audit Bus PDP
91. SSO ses act	$t+4S+4X=8$	$4T+3S+5X=14$	$2T+2S+3X+Z=8.5$			$4(2T+S+3X)=28$	$4(t+2X+Z)=10$
92. Discovery A	$2t+3S+3X=6$	$T+S+X=3.5$				$2t+S+3X=4$	$t+2X+Z=2.5$
93. Trust & Priv.	$T+2X=3.5$				$2T+S+3X=7$	$2T+S+3X=7$	$t+2X+Z=2.5$
94. Rq Out PEP		$t+2X=2$	$2t+2S+4X+1Z=6.5$			$2t+S+3X=4$	$t+2X+Z=2.5$
95. Send request		$T+t+2S+2X=5.5$		$T+t+3S+3X=7.5$		$2(2t+S+3X)=8$	$2(t+2X+Z)=5$
96. Rs In PEP				$T+2X=3.5$	$2T+2S+4X+Z=9.5$	$2t+S+3X=4$	$t+2X+Z=2.5$
97. Payload							
98. Rs Out PEP				$t+2X=2$	$2t+2S+4X+Z=6.5$	$2t+S+3X=4$	$t+2X+Z=2.5$
99. Send respons		$t+2S+2X=4$		$t+2S+2X=4$		$2(2t+S+3X)=8$	$2(t+2X+Z)=5$
100 Rq In PEP		$t+2X=2$	$2t+2S+4X+Z=6.5$			$2t+S+3X=4$	$t+2X+Z=2.5$
101 Process Obli		$2t+S+2X=3$		$2t+S+2X=3$		$2(2t+S+3X)=8$	$2(t+2X+Z)=5$
102 Discovery D	$2t+3S+3X=6$	$T+S+X=3.5$				$2t+S+3X=4$	$t+2X+Z=2.5$
103 Trust & Priv.	$T+2X=3.5$				$2T+S+3X=7$	$2T+S+3X=7$	$t+2X+Z=2.5$
104 Rq Out PEP		$t+2X=2$	$2t+2S+4X+1Z=6.5$			$2t+S+3X=4$	$t+2X+Z=2.5$
105 Send request		$T+t+2S+2X=5.5$		$T+t+3S+3X=7.5$		$2(2t+S+3X)=8$	$2(t+2X+Z)=5$
106 Rs In PEP				$T+2X=3.5$	$2T+2S+4X+Z=9.5$	$2t+S+3X=4$	$t+2X+Z=2.5$
107 Payload							
108 Rs Out PEP				$t+2X=2$	$2t+2S+4X+Z=6.5$	$2t+S+3X=4$	$t+2X+Z=2.5$
109 Send respons		$t+2S+2X=4$		$t+2S+2X=4$		$2(2t+S+3X)=8$	$2(t+2X+Z)=5$
110 Rq In PEP		$t+2X=2$	$2t+2S+4X+Z=6.5$			$2t+S+3X=4$	$t+2X+Z=2.5$
111 Process Obli		$2t+S+2X=3$		$2t+S+2X=3$		$2(2t+S+3X)=8$	$2(t+2X+Z)=5$
112 Rq Out PEP		$t+2X=2$	$2t+2S+4X+1Z=6.5$			$2t+S+3X=4$	$t+2X+Z=2.5$
113 Send request		$2t+2S+2X=4$		$2t+3S+3X=6$		$2(2t+S+3X)=8$	$2(t+2X+Z)=5$
114 Rs In PEP				$t+2X=2$	$2t+2S+4X+Z=6.5$	$2t+S+3X=4$	$t+2X+Z=2.5$
115 Payload							
116 Rs Out PEP				$t+2X=2$	$2t+2S+4X+Z=6.5$	$2t+S+3X=4$	$t+2X+Z=2.5$
117 Send respons		$t+2S+2X=4$		$t+2S+2X=4$		$2(2t+S+3X)=8$	$2(t+2X+Z)=5$
118 Rq In PEP		$t+2X=2$	$2t+2S+4X+Z=6.5$			$2t+S+3X=4$	$t+2X+Z=2.5$
119 Process Obli		$2t+S+2X=3$		$2t+S+2X=3$		$2(2t+S+3X)=8$	$2(t+2X+Z)=5$
120 Rq Out PEP		$t+2X=2$	$2t+2S+4X+1Z=6.5$			$2t+S+3X=4$	$t+2X+Z=2.5$
121 Send request		$2t+2S+2X=4$		$2t+3S+3X=6$		$2(2t+S+3X)=8$	$2(t+2X+Z)=5$
122 Rs In PEP				$t+2X=2$	$2t+2S+4X+Z=6.5$	$2t+S+3X=4$	$t+2X+Z=2.5$
123 Payload							
124 Rs Out PEP				$t+2X=2$	$2t+2S+4X+Z=6.5$	$2t+S+3X=4$	$t+2X+Z=2.5$
125 Send respons		$t+2S+2X=4$		$t+2S+2X=4$		$2(2t+S+3X)=8$	$2(t+2X+Z)=5$
126 Rq In PEP		$t+2X=2$	$2t+2S+4X+Z=6.5$			$2t+S+3X=4$	$t+2X+Z=2.5$
127 Process Obli		$2t+S+2X=3$		$2t+S+2X=3$		$2(2t+S+3X)=8$	$2(t+2X+Z)=5$
128 Rq Out PEP		$t+2X=2$	$2t+2S+4X+1Z=6.5$			$2t+S+3X=4$	$t+2X+Z=2.5$
129 Send request		$2t+2S+2X=4$		$2t+3S+3X=6$		$2(2t+S+3X)=8$	$2(t+2X+Z)=5$
130 Rs In PEP				$t+2X=2$	$2t+2S+4X+Z=6.5$	$2t+S+3X=4$	$t+2X+Z=2.5$
131 Payload							
132 Rs Out PEP				$t+2X=2$	$2t+2S+4X+Z=6.5$	$2t+S+3X=4$	$t+2X+Z=2.5$
133 Send respons		$t+2S+2X=4$		$t+2S+2X=4$		$2(2t+S+3X)=8$	$2(t+2X+Z)=5$
134 Rq In PEP		$t+2X=2$	$2t+2S+4X+Z=6.5$			$2t+S+3X=4$	$t+2X+Z=2.5$
135 Process Obli		$2t+S+2X=3$		$2t+S+2X=3$		$2(2t+S+3X)=8$	$2(t+2X+Z)=5$
136 SLO	$2T+2S+3X=8$	$2T+2S+3X=8$				$2(2t+S+3X)=8$	$2(T+2X+Z)=8$
TOTAL	$10T+32S+45X$	$26T+92S+174X$	$6T+66S+129X+33Z$	$12T+90S+165X$	$24T+66S+138X+30Z$	$236T+176S+528X$	$T+352X+176Z$
TOTAL RSA	=92	=305	=220.5	=273	=255	=758	=443

2062

2063

2064

2065

2066

2067

This sequence of 15 web service calls has grand total of  $116T+522S+1531X+239Z=2346.5$  RSA equivalents, which works out to about 156 RSA equivalents per web service call. As can be seen the cache effects and amortization of the SSO and discovery over several calls makes a significant impact. The amortized cost is 58% of the single call cost. Effectively the amortized calls are 18 times heavier than plain web service calls.

2068

2069

## 7 Annex A: Examples

2070

2071

2072

These XML blobs, taken from [ZXIDREADME], are for reference only. They are not normative. They have been pretty printed. Indentation indicates nesting level and closing tags have been abbreviated as "</>". The actual XML on the wire generally does not have any whitespace.

2074

2075

### 7.1 SAML 2.0 Artifact Response with SAML 2.0 SSO Assertion and Two Bootstraps

2076

Both bootstraps illustrate SAML assertion as bearer token.

2077

2078

2079

2080

2081

2082

2083

2084

2085

2086

2087

2088

2089

2090

2091

2092

2093

2094

2095

2096

2097

2098

2099

2100

2101

2102

2103

2104

2105

2106

2107

2108

2109

2110

2111

2112

2113

2114

2115

2116

2117

2118

```

<soap:Envelope
  xmlns:lib="urn:liberty:iff:2003-08"
  xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:wsa="http://www.w3.org/2005/08/addressing">
  <soap:Body>

    <sp:ArtifactResponse
      xmlns:sp="urn:oasis:names:tc:SAML:2.0:protocol"
      ID="REvgoIilkzTmk-aIX6tKE"
      InResponseTo="RfAsltVf2"
      IssueInstant="2007-02-10T05:38:15Z"
      Version="2.0">
      <sa:Issuer
        xmlns:sa="urn:oasis:names:tc:SAML:2.0:assertion"
        Format="urn:oasis:names:tc:SAML:2.0:nameid-format:entity">
        https://a-idp.liberty-iop.org:8881/idp.xml</>
      <sp:Status>
        <sp:StatusCode Value="urn:oasis:names:tc:SAML:2.0:status:Success"/></>

      <sp:Response
        xmlns:sp="urn:oasis:names:tc:SAML:2.0:protocol"
        ID="RCCzu13z77SiSXqsFplu1"
        InResponseTo="NojFIihxw"
        IssueInstant="2007-02-10T05:37:42Z"
        Version="2.0">
        <sa:Issuer
          xmlns:sa="urn:oasis:names:tc:SAML:2.0:assertion"
          Format="urn:oasis:names:tc:SAML:2.0:nameid-format:entity">
          https://a-idp.liberty-iop.org:8881/idp.xml</>
        <sp:Status>
          <sp:StatusCode Value="urn:oasis:names:tc:SAML:2.0:status:Success"/></>

        <sa:Assertion
          xmlns:sa="urn:oasis:names:tc:SAML:2.0:assertion"
          ID="ASSE6bgfaV-sapQsAilXOvBu"
          IssueInstant="2007-02-10T05:37:42Z"
          Version="2.0">
          <sa:Issuer Format="urn:oasis:names:tc:SAML:2.0:nameid-format:entity">
            https://a-idp.liberty-iop.org:8881/idp.xml</>

          <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
            <ds:SignedInfo>
    
```

```

2119     <ds:CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" />
2120     <ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1" />
2121     <ds:Reference URI="#ASSE6bgfaV-sapQsAilXOvBu">
2122         <ds:Transforms>
2123             <ds:Transform Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature" />
2124             <ds:Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" /></>
2125             <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
2126             <ds:DigestValue>r8OvtNmq5LkYwCNg6bsRZAdT4NE=</></></>
2127         <ds:SignatureValue>GtWVZzHYW54ioHk/C7zjDRThohrpwC4=</></>
2128
2129     <sa:Subject>
2130         <sa:NameID
2131             Format="urn:oasis:names:tc:SAML:2.0:nameid-format:persistent"
2132             NameQualifier="https://a-idp.liberty-iop.org:8881/idp.xml">PB5fLIA41RU2bH4HkQsn
2133         <sa:SubjectConfirmation
2134             Method="urn:oasis:names:tc:SAML:2.0:cm:bearer">
2135             <sa:SubjectConfirmationData
2136                 NotOnOrAfter="2007-02-10T06:37:41Z"
2137                 Recipient="https://spl.zxidsp.org:8443/zxidhlo?o=B" /></></>
2138
2139         <sa:Conditions
2140             NotBefore="2007-02-10T05:32:42Z"
2141             NotOnOrAfter="2007-02-10T06:37:42Z">
2142         <sa:AudienceRestriction>
2143             <sa:Audience>https://spl.zxidsp.org:8443/zxidhlo?o=B</></></>
2144
2145     <sa:Advice>
2146
2147     <!-- This assertion is the credential for the ID-WSF 1.1 bootstrap (below). -->
2148
2149     <sa:Assertion
2150         ID="CREDOTGakvhNoPlaiTq4bXBg"
2151         IssueInstant="2007-02-10T05:37:42Z"
2152         Version="2.0">
2153         <sa:Issuer
2154             Format="urn:oasis:names:tc:SAML:2.0:nameid-format:entity">
2155             https://a-idp.liberty-iop.org:8881/idp.xml</>
2156         <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
2157             <ds:SignedInfo>
2158                 <ds:CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" />
2159                 <ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1" />
2160                 <ds:Reference URI="#CREDOTGakvhNoPlaiTq4bXBg">
2161                     <ds:Transforms>
2162                         <ds:Transform Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature" />
2163                         <ds:Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" /></>
2164                         <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
2165                         <ds:DigestValue>dqQ/28hw5eEv+ceFyiLlmeJ1P8w=</></></>
2166                     <ds:SignatureValue>UKlEgHKQwuoCE=</></>
2167                 <sa:Subject>
2168                     <sa:NameID/> <!-- *** Bug here!!! -->
2169                     <sa:SubjectConfirmation
2170                         Method="urn:oasis:names:tc:SAML:2.0:cm:bearer" /></>
2171                 <sa:Conditions

```

```

2172         NotBefore="2007-02-10T05:32:42Z"
2173         NotOnOrAfter="2007-02-10T06:37:42Z">
2174     <sa:AudienceRestriction>
2175         <sa:Audience>https://sp1.zxidsp.org:8443/zxidhlo?o=B</></></></></>
2176
2177     <sa:AuthnStatement
2178         AuthnInstant="2007-02-10T05:37:42Z"
2179         SessionIndex="1171085858-4">
2180     <sa:AuthnContext>
2181         <sa:AuthnContextClassRef>
2182             urn:oasis:names:tc:SAML:2.0:ac:classes>Password</></></>
2183
2184     <sa:AttributeStatement>
2185
2186     <!-- Regular attribute -->
2187
2188     <sa:Attribute
2189         Name="cn"
2190         NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:basic">
2191     <sa:AttributeValue>Sue</></>
2192
2193     <!-- ID-WSF 1.1 Bootstrap for discovery. See also the Advice, above. -->
2194
2195     <sa:Attribute
2196         Name="DiscoveryResourceOffering"
2197         NameFormat="urn:liberty:disco:2003-08">
2198     <sa:AttributeValue>
2199         <di12:ResourceOffering
2200             xmlns:di12="urn:liberty:disco:2003-08"
2201             entryID="2">
2202         <di12:ResourceID>
2203             https://a-idp.liberty-iop.org/profiles/WSF1.1/RID-DISCO-sue</>
2204         <di12:ServiceInstance>
2205             <di12:ServiceType>urn:liberty:disco:2003-08</>
2206             <di12:ProviderID>https://a-idp.liberty-iop.org:8881/idp.xml</>
2207             <di12:Description>
2208                 <di12:SecurityMechID>urn:liberty:security:2005-02:TLS:Bearer</>
2209                 <di12:CredentialRef>CREDOTGAKvhNoPlaiTq4bXBg</>
2210                 <di12:Endpoint>https://a-idp.liberty-iop.org:8881/DISCO-S</></></>
2211                 <di12:Abstract>Symlabs Discovery Service Team G</></></></>
2212
2213     <!-- ID-WSF 2.0 Bootstrap for Discovery. The credential (bearer token) is inline. -->
2214
2215     <sa:Attribute
2216         Name="urn:liberty:disco:2006-08:DiscoveryEPR"
2217         NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:uri">
2218     <sa:AttributeValue>
2219         <wsa:EndpointReference
2220             xmlns:wsa="http://www.w3.org/2005/08/addressing"
2221             xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity"
2222             notOnOrAfter="2007-02-10T07:37:42Z"
2223             wsu:Id="EPRIDcjP80b09In47SDj09b37">
2224         <wsa:Address>https://a-idp.liberty-iop.org:8881/DISCO-S</>

```

```

2225 <wsa:Metadata xmlns:di="urn:liberty:disco:2006-08">
2226   <di:Abstract>SYMfiam Discovery Service</di>
2227   <sbef:Framework xmlns:sbf="urn:liberty:sb" version="2.0"/>
2228   <di:ProviderID>https://a-idp.liberty-iop.org:8881/idp.xml</di>
2229   <di:ServiceType>urn:liberty:disco:2006-08</di>
2230   <di:SecurityContext>
2231     <di:SecurityMechID>urn:liberty:security:2005-02:TLS:Bearer</di>
2232
2233   <sec:Token
2234     xmlns:sec="urn:liberty:security:2006-08"
2235     usage="urn:liberty:security:tokenusage:2006-08:SecurityToken">
2236
2237     <sa:Assertion
2238       ID="CREDV6ZBMyicmyvDq9pLIoSR"
2239       IssueInstant="2007-02-10T05:37:42Z"
2240       Version="2.0">
2241       <sa:Issuer Format="urn:oasis:names:tc:SAML:2.0:nameid-format:entity">
2242         https://a-idp.liberty-iop.org:8881/idp.xml</sa:Issuer>
2243       <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
2244         <ds:SignedInfo>
2245           <ds:CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml
2246             <ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#
2247             <ds:Reference URI="#CREDV6ZBMyicmyvDq9pLIoSR">
2248               <ds:Transforms>
2249                 <ds:Transform Algorithm="http://www.w3.org/2000/09/xmldsig#en
2250                 <ds:Transform Algorithm="http://www.w3.org/2001/10/xml-exc-cl
2251                 <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sh
2252                 <ds:DigestValue>o2SgbuKIBz14e0dQoTwiqXr/8Y=</ds:DigestValue>
2253                 <ds:SignatureValue>hHdUKaZ//cZ8UYJxvTReNU=</ds:SignatureValue>
2254             <sa:Subject>
2255               <sa:NameID
2256                 Format="urn:oasis:names:tc:SAML:2.0:nameid-format:persistent"
2257                 NameQualifier="https://a-idp.liberty-iop.org:8881/idp.xml">
2258                 9my93VkP3tSxE0Ib3ckvjLpn0pa6aV3yFXioWX-TzZI=</sa:NameID>
2259             <sa:SubjectConfirmation
2260               Method="urn:oasis:names:tc:SAML:2.0:cm:bearer"/></sa:SubjectConfirmation>
2261             <sa:Conditions
2262               NotBefore="2007-02-10T05:32:42Z"
2263               NotOnOrAfter="2007-02-10T06:37:42Z">
2264               <sa:AudienceRestriction>
2265                 <sa:Audience>https://a-idp.liberty-iop.org:8881/idp.xml</sa:Audience>
2266             <sa:AuthnStatement AuthnInstant="2007-02-10T05:37:42Z">
2267               <sa:AuthnContext>
2268                 <sa:AuthnContextClassRef>
2269                 urn:oasis:names:tc:SAML:2.0:ac:classes:Password</sa:AuthnContextClassRef>

```

2270 N.B. The AttributeStatement/Attribute/AttributeValue/EndpointReference/Metadata/ SecurityContext  
2271 is the same as the IdP because in many products the IdP and Discovery Service roles are implemented by  
2272 the same entity. Note also that the audience of the inner assertion is the discovery service where as the  
2273 audience of the outer assertion is the SP that will eventually call the Discovery Service.

## 2275 7.2 ID-WSF 2.0 Call with X509v3 Sec Mech

```
2276 <e:Envelope
```

```

2277     xmlns:e="http://schemas.xmlsoap.org/soap/envelope/"
2278     xmlns:b="urn:liberty:sb:2005-11"
2279     xmlns:sec="urn:liberty:security:2005-11"
2280     xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.x
2281     xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.x
2282     xmlns:wsa="http://www.w3.org/2005/08/addressing">
2283 <e:Header>
2284   <wsa:MessageID wsu:Id="MID">123</>
2285   <wsa:To wsu:Id="TO">...</>
2286   <wsa:Action wsu:Id="ACT">urn:xx:Query</>
2287   <wsse:Security mustUnderstand="1">
2288     <wsu:Timestamp wsu:Id="TS"><wsu:Created>2005-06-17T04:49:17Z</></>
2289     <wsse:BinarySecurityToken
2290       ValueType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile
2291       wsu:Id="X509Token"
2292       EncodingType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-s
2293       MIIB9zCCAWSgAwIBAgIQ...</>
2294     <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
2295       <ds:SignedInfo>
2296         <ds:Reference URI="#MID">...</>
2297         <ds:Reference URI="#TO">...</>
2298         <ds:Reference URI="#ACT">...</>
2299         <ds:Reference URI="#TS">...</>
2300         <ds:Reference URI="#X509">
2301           <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
2302           <ds:DigestValue>Ru4cAfeBAB</></>
2303         <ds:Reference URI="#BDY">
2304           <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
2305           <ds:DigestValue>YgGfS0pi56p</></></>
2306         <ds:KeyInfo><wsse:SecurityTokenReference><wsse:Reference URI="#X509"/></></>
2307         <ds:SignatureValue>HJJWbvqW9E84vJVQkjDElgscSXZ5Ekw==</></></></>
2308     </e:Body wsu:Id="BDY">
2309       <xx:Query/></></>

```

2310 The salient features of the above XML blob are

- 2311 • Signature that covers relevant SOAP headers and Body
- 2312 • Absence of any explicit identity token.

2313 Absence of identity token means that from the headers it is not possible to identify the target identity.  
 2314 The signature generally covers the Invoker identity (the WSC that is calling the service). Since one WSC  
 2315 typically serves many principals, knowing which principal is impossible. For this reason X509 security  
 2316 mechanism is seldom used in ID-WSF 2.0 world (with ID-WSF 1.1 the ResourceID provides an alternative  
 2317 way of identifying the principal, thus making X509 a viable option).

### 2319 7.3 ID-WSF 2.0 Call with Bearer (Binary) Sec Mech

```

2320 <e:Envelope
2321   xmlns:e="http://schemas.xmlsoap.org/soap/envelope/"
2322   xmlns:b="urn:liberty:sb:2005-11"
2323   xmlns:sec="urn:liberty:security:2005-11"
2324   xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.x
2325   xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.x

```

```

2326     xmlns:wsa="http://www.w3.org/2005/03/ addressing">
2327 <e:Header>
2328     <wsa:MessageID wsu:Id="MID">...</>
2329     <wsa:To wsu:Id="TO">...</>
2330     <wsa:Action wsu:Id="ACT">urn:xx:Query</>
2331     <wsse:Security mustUnderstand="1">
2332         <wsu:Timestamp wsu:Id="TS">
2333             <wsu:Created>2005-06-17T04:49:17Z</></>
2334         <wsse:BinarySecurityToken
2335             ValueType="anyNSPrefix:ServiceSessionContext"
2336             EncodingType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-se
2337             wsu:Id="BST">
2338             mQEMAzRniWkAAAEH9RWir0eKDkyFAB7PoFazx3ftp0vWwbbzqXdgcX8fpEqSrlv4
2339             YqUc70MiJcBtKBp3+jlD4HPUaurIqHA0vrDmMpM+sF2BnpND118f/mXCv3XbWhiL
2340             VT4r9ytfpXBlue1OV93X8RUz4ecZcDm9e+IEG+pQjnvgrSgac1NrW5K/CJEOUJh
2341             oGTrym0Ziutezhrw/g0eLVtkywsMgDr77gWZxRvw01wlogtUdTceurBIDANj+KVZ
2342             vLKlTCaGAUNIjkiDDgti=</>
2343         <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig #">
2344             <ds:SignedInfo>
2345                 <ds:Reference URI="#MID">...</>
2346                 <ds:Reference URI="#TO">...</>
2347                 <ds:Reference URI="#ACT">...</>
2348                 <ds:Reference URI="#TS">...</>
2349                 <ds:Reference URI="#BST">...</>
2350                 <ds:Reference URI="#BDY">
2351                     <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1 " />
2352                     <ds:DigestValue>YgGfS0pi56pu</></></>
2353                 ...</></></>
2354             <e:Body wsu:Id="BDY">
2355                 <xx:Query/></></>
2356

```

## 2357 7.4 ID-WSF 2.0 Call with Bearer (SAML) Sec Mech

```

2358 <e:Envelope
2359     xmlns:e="http://schemas.xmlsoap.org/soap/envelope/"
2360     xmlns:sb="urn:liberty:sb:2005-11"
2361     xmlns:sec="urn:liberty:security:2005-11"
2362     xmlns:wsse="http://docs.oasis-open.org/wss/20 04/01/oasis-200401-wss-wssecurity-secext-1.0.
2363     xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.x
2364     xmlns:wsa="http://www.w3.org/2005/08/addressing"
2365     xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
2366     xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
2367 <e:Header>
2368     <sbf:Framework version="2.0-simple" e:mustUnderstand="1"
2369         e:actor="http://schemas.../next"
2370         wsu:Id="SBF"/>
2371     <wsa:MessageID wsu:Id="MID">...</>
2372     <wsa:To wsu:Id="TO">...</>
2373     <wsa:Action wsu:Id="ACT">urn:xx:Query</>
2374     <wsse:Security mustUnderstand="1">
2375         <wsu:Timestamp wsu:Id="TS">
2376             <wsu:Created>2005-06-17T04:49:17Z</></>
2377

```

```

2378     <sa:Assertion
2379         xmlns:sa="urn:oasis:names:tc:SAML:2.0:assertion"
2380         Version="2.0"
2381         ID="A7N123"
2382         IssueInstant="2005-04-01T16:58:33.173Z">
2383     <sa:Issuer>http://idp.symdemo.com/idp.xml</>
2384     <ds:Signature>...</>
2385     <sa:Subject>
2386         <sa:EncryptedID>
2387             <xenc:EncryptedData>U2XTCNvRX7B11NK182nmY00TEk==</>
2388             <xenc:EncryptedKey>...</></>
2389         <sa:SubjectConfirmation Method="urn:oasis:names:tc:SAML:2.0:cm:bearer"/></>
2390     <sa:Conditions>
2391         NotBefore="2005-04-01T16:57:20Z"
2392         NotOnOrAfter="2005-04-01T21:42:4 3Z">
2393         <sa:AudienceRestrictionCondition>
2394             <sa:Audience>http://wsp.zxidsp.org</></></>
2395     <sa:AuthnStatement>
2396         AuthnInstant="2005-04-01T16:57:30.000Z"
2397         SessionIndex="6345789">
2398     <sa:AuthnContext>
2399         <sa:AuthnContextClassRef>
2400             urn:oasis:names:tc:SAML:2.0:ac:classes>PasswordProtectedTransport</></></>
2401     <sa:AttributeStatement>
2402         <sa:EncryptedAttribute>
2403             <xenc:EncryptedData Type="http://www.w3.org/2001/04/xmlenc#Element">
2404                 mQEMAZRniWkAAAEH9RbzqXdgcX8fpEqSrlv4=</>
2405             <xenc:EncryptedKey>...</></></></>
2406
2407     <wsse:SecurityTokenReference
2408         xmlns:wsse1="..."
2409         wsu:Id="STR1"
2410         wsse1:TokenType="http://docs.oasis-open.org/wss/oasis-wss-saml-token-profile-1.1#SAMLID"
2411     <wsse:KeyIdentifier
2412         ValueType="http://docs.oasis-open.org/wss/oasis-wss-saml-token-profile-1.1#SAMLID">
2413         A7N123</></>
2414
2415     <ds:Signature>
2416         <ds:SignedInfo>
2417             <ds:Reference URI="#MID">...</>
2418             <ds:Reference URI="#T0">...</>
2419             <ds:Reference URI="#ACT">...</>
2420             <ds:Reference URI="#TS">...</>
2421             <ds:Reference URI="#STR1">
2422                 <ds:Transform Algorithm="...#STR-Transform">
2423                     <wsse:TransformationParameters>
2424                         <ds:CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20011116#XML11">
2425                             <ds:Reference URI="#BDY"/></>
2426                         ...</></></>
2427         </ds:SignedInfo>
2428     </ds:Signature>
    </e:Body wsu:Id="BDY">
    <xx:Query/></></>

```

2429 Note how the <Subject> and the attributes are encrypted such that only the WSP can open them. This  
2430 protects against WSC gaining knowledge of the NameID at the WSP.

2431

2432

2433

## 8 Annex B: Technical Self Assessment Questionnaire

2434

2435

2436

2437

This questionnaire is to be used in partner intake process of a TAS<sup>3</sup> compliant Trust Network. Effectively this is a template that the trust network can adjust corresponding to its own policies. Typically this questionnaire is used along side the legal questionnaire, see [TAS3D62Contract], 11.6 Annex IV "Self Assessment Questionnaire".

2439

### 8.1 Overview and Scope

2440

1. Please give your installation a unique name or reference that can be used in future communications.

2441

Installation Name: \_\_\_\_\_

2442

2. Please supply your organizational and contact details

2443

\_\_\_\_\_

2444

\_\_\_\_\_

2445

\_\_\_\_\_

2446

Technical contact for clarifications: \_\_\_\_\_

2447

Who filled this questionnaire: \_\_\_\_\_

2448

Date when filled or amended: \_\_\_\_\_

2449

3. What architectural roles do you plan to play in Trust Network? (tick all that apply)

2450

a.  Service Provider (SP), such as Frontend Web Site (FE), Web Services Client (WSC), Web Services Provider (WSP) (other than WSP acting as Attribute Authority, see below).

2451

2452

b.  Attribute or Credentials Authority as a web service (some people call attribute authorities also "identity providers", but see next item if you are performing SSO)

2453

2454

c.  Single Sign-On Identity Provider, Discovery Service, Discovery Registry, Identity Mapper, or Delegation Service.

2455

2456

d.  Identity Aggregator or Linking Service

2457

e.  Authorization Supplier (e.g. PDP) or Ontology Mapper towards external parties (if you merely operate PDP internally, you do not need to tick this)

2458

2459

f.  Trust and Reputation provider towards external parties

2460

g.  User Audit Dashboard or Interaction Service provider; or Credentials and Privacy Negotiation agent for the user

2461

2462

h.  Online Compliance Testing Provider

2463

i.  Trust Network configuration, management, oversight, or audit services; or certification authority.

2464

2465

j.  Other, please specify: \_\_\_\_\_

2466

4. For each of the service instances you plan to run, please provide domain names and EntityIDs. If not known yet, specify "not yet assigned" or "NYA".

2467

2468

Extend the table as needed or provide annex (e.g. spreadsheet with the information).

2469

This table is just an initial survey and it is understood that it can be amended from time to time.

2470

5. How do you plan to implement the service instances?

Table 8.1: Basic information about entities

N	Domain Name	EntityID	Roles	Remarks
1.	sp.example.com	https://sp.example.com/svc?o=B	FE, WSC	Example SP entry
2.				
3.				

2471 a.  Complete outsource to a partner, which: \_\_\_\_\_

2472

2473 If you tick this box you should have the partner fill the technical details of this questionnaire, or  
2474 provide a reference to a questionnaire they have filled separately.

2475 b.  Software as a Service (SaaS), operated by you.

2476 Which software or partner: \_\_\_\_\_, version: \_\_\_\_

2477 Your SaaS provider should help you answer the technical questions.

2478 c.  Operate commercial software on servers administered by you (e.g. own server, hosted root  
2479 server, server on Amazon Elastic Cloud, etc.)

2480 Which software: \_\_\_\_\_, version: \_\_\_\_

2481 d.  Operate open source software on servers administered by you (e.g. own server, hosted root  
2482 server, server on Amazon Elastic Cloud, etc.)

2483 Which software: \_\_\_\_\_, version: \_\_\_\_

2484 e.  Operate software developed by you or for you

2485 Which software: \_\_\_\_\_, version: \_\_\_\_

2486 6. Please provide volumetrics about your installation. We realize some of this information may not be  
2487 public or may not be available or accurate. Any information you can provide is helpful.

2488 Number of potential users: \_\_\_\_\_

2489 Number of regular or frequent users: \_\_\_\_\_

2490 Number of tasks performed by a regular user on typical working day on your service: \_\_\_\_\_

2491 Any performance targets you expect from the system, such as maximum latency or required throughput:  
2492 \_\_\_\_\_

2493 7. Do you plan to implement any load balancing, scaling, or redundant resiliency measures? Please  
2494 specify: \_\_\_\_\_

2495

## 2496 8.2 System Entity Credentials and Private Keys

2497 In TAS<sup>3</sup>, services and other system entities are identified using X509 digital certificates. They are used  
2498 in TLS connections for authentication using Client TLS and they are used for digital signatures.

2499 Responsible management of the private keys associated with the digital certificates is the corner stone  
2500 of TAS<sup>3</sup> accountability and liability framework. Your organization will be held responsible for all actions  
2501 performed using your private keys.

2502 1. Which certification authority do you use for issuance of certificates? (if selfissued, indicate who in  
2503 your organization is responsible)

2504 \_\_\_\_\_

2505 2. How do you generate private key and certification request?

2506 \_\_\_\_\_

- 2507 3. What measures are in place to ensure that the private key remains confidential during generation, cer-  
 2508 tificate issuance, and installation process? How do you know that no copy is left on any device (e.g.  
 2509 USB stick of a consultant) used to handle the private key?  
 2510 \_\_\_\_\_
- 2511 4. What backup arrangements do you have for the private key and how are they kept confidential?  
 2512 \_\_\_\_\_
- 2513 5. Once installed on a server, how do you ensure confidentiality of the private key? (tick all that apply)
- 2514 a.  Private key protected by hardware token  
 2515 b.  Password required for each use of private key  
 2516 c.  Password required for first use after reboot  
 2517 d.  Filesystem permissions  
 2518 e.  No root or administration access over the network. For example if you have configured *sudo(8)*  
 2519 so that no user is unlimited root and only appropriate process has access to the private key.  
 2520 f.  All system administrators are authorized to access the private key  
 2521 g. Other: \_\_\_\_\_
- 2522 6. If private key could be stored in a jump start, kick start, or backup image, what confidentiality measures  
 2523 are in place to protect such images? \_\_\_\_\_
- 2524 7. Do you track or register who is authorized to access private keys?  
 2525 How: \_\_\_\_\_  
 2526 Are there written records? \_\_\_\_\_
- 2527 8. Do you track or register who has system administration access to servers, especially if not all sysadms  
 2528 are authorized to access private keys?
- 2529 9. Do all those who are authorized to access private keys or who could have access to the private keys  
 2530 (e.g. sysadms) go through training on private keys and sign a confidentiality undertaking regarding  
 2531 them? \_\_\_\_\_  
 2532

### 2533 8.3 Trust Management

- 2534 1. What is your organization's policy regarding which entities to trust:
- 2535 a.  Trust anyone  
 2536 b.  Trust all members of the Trust Network  
 2537 c.  Trust all members of the Trust Network that also pass local check (e.g. black list)  
 2538 d.  Explicit local check (e.g. white list)  
 2539 e.  Other, please describe: \_\_\_\_\_
- 2540 2. What administrative and system administration procedures do you have in place to check that your  
 2541 software is configured to trust only the entities that your organization has decided to trust?
- 2542 3. What techniques and procedures do you use to ensure that the trust settings are not tampered with and  
 2543 that if tampered, you detect the alterations in a timely manner?

2544

## 2545 8.4 Threat and Risk Assessments

- 2546 1. Have you reviewed TAS<sup>3</sup> Threat Analysis document [[TAS3THREAT](#)]?  
 2547 2. Have you reviewed TAS<sup>3</sup> Risk Assessment document [[TAS3RISK](#)]?  
 2548 3. With respect to the services you plan to deploy, which of the mitigation techniques discussed in  
 2549 [[TAS3RISK](#)] do you plan to implement?  
 2550

## 2551 8.5 Service Provider Questions

- 2552 1. What is your Entity ID? \_\_\_\_\_  
 2553 Entity ID is decided by you, the organization operating the service. It should be a URL pointing to  
 2554 your SAML metadata. Typically it consists of your domain name, some local path, and possibly of  
 2555 software package dependent part. For example, in

2556 `https://sp.example.com/svc?o=B`

2557 the domain name is "sp.example.com", the local path is "/svc" and the product dependent part is  
 2558 "?o=B". The local path depends on how your web server is configured. Consult product documen-  
 2559 tation for the product dependent part, if any.

- 2560 2. Does your site support Well Known Location method of SAML metadata exchange (i.e. the metadata  
 2561 is available in the Entity ID URL, consult product documentation if in doubt)?  
 2562 () Yes, () No  
 2563 If not, what alternative arrangements do you have for metadata exchange?  
 2564 3. How do you provide audit drilldown? (check all that apply)  
 2565 a. () Stand alone web GUI. URL: \_\_\_\_\_  
 2566 b. () iFrame widget Web GUI. URL: \_\_\_\_\_  
 2567 c. () Audit drill down web service (ServiceType "urn:tas3:audit:2010-06")  
 2568 4. Have you successfully tested sending messages to the Audit Event Bus?  
 2569

### 2570 8.5.1 Front End (FE) Single Sign-On Questions

- 2571 1. Is your software SAML 2.0 compliant? Is it certified? When, by whom: \_\_\_\_  
 2572 2. Can your software handle ID-WSF 2.0 discovery bootstrap?  
 2573 3. Which IdPs do you plan to use?  
 2574 4. Have you exchanged metadata with the IdP?  
 2575 5. Have you successfully tested SSO with the IdP?

2576

2577 **8.5.2 Web Service Provider (WSP) Questions**

2578 1. Is your software TAS<sup>3</sup> or ID-WSF 2.0 compliant?

2579 Is it certified? When, by whom: \_\_\_\_\_

2580 2. Have you determined

2581 a. SOAP endpoint URL: \_\_\_\_\_

2582 b. Human friendly name for your service: \_\_\_\_\_

2583 c. Entity ID of your service (usually different from SOAP endpoint): \_\_\_\_\_

2584 d. Service Type URI of your service: \_\_\_\_\_

2585 The Service Type URI designates the type of service you provide. If you are providing a standard-  
2586 ized service, the relevant standard should specify what the Service Type URI is for services of that  
2587 type. All instances of the service use the same Service Type URI. Some well known Service Types:

- 2588 • "urn:ios:pds:2010-05:dst-2.1" - Internet of Subjects Personal Data Store
- 2589 • "urn:liberty:id-sis-dap:2006-08:dst-2.1" - Liberty ID Directory Access Protocol
- 2590 • "urn:liberty:id-sis-cb:2004-10" - Liberty Contact Book Service
- 2591 • "urn:liberty:id-sis-gl:2005-07" - Liberty Geolocation Service
- 2592 • "http://www.3gpp.org/ftp/Specs/archive/23\_series/23.140/schema/REL-6-MM7-1-4"  
2593 - ID-MM7 messaging service

2594 If you created the service yourself, you can pick the URI as you please, provided that it is globally  
2595 unique. The usual convention is to use the namespace URI of the top level XML element of the  
2596 service payload, i.e. the namespace of the first child element of SOAP Envelope Body element.

2597 3. Have you registered your service end point with a Discovery Service?

2598 Often the Discovery Service Provider or IdP provides a registration interface on the web. For example  
2599 the TAS<sup>3</sup> IdP provides "Circle of Trust Manager" at URL <https://idp.tas3.eu/cot/>

2600 If you do not plan to use discovery, what arrangements do you plan to use to locate your service? What  
2601 arrangements do you plan to make for issuing security tokens for accessing your service?

2602 4. Have you successfully tested calling your web service from a third party web service client?

2603 5. Is your service an identity service, i.e. does it need to know something about the user?

2604 6. Does your service need persistent handle to user, e.g. to track something about the user (this question  
2605 aims to establish whether your service needs to see persistent or transient NameID)?

2606 7. What types of credentials need to be presented upon web service call to authorize the call?

2607 This question aims at determining what credentials your callers will need to gather and present. We do  
2608 not need full description of your policy.

2609 8. Do you need user to consent to anything and how do you arrange to obtain consent when needed? Do  
2610 you plan to use the Interaction Service facility and/or handle Interaction Redirect?

2611 9. Are you capable to act as a Credentials and Privacy Negotiation server? If yes, please provide end point  
2612 URL: \_\_\_\_\_

2613 10. What security mechanisms are you willing and able to support

2614 a. (  ) Bearer Token

2615 b. (  ) Holder of Key Token

- 2616 c.  X509 signature without token  
 2617 d.  None
- 2618 11. Which Policy Enforcement Points do you implement?
- 2619 a.  Request Out PEP  
 2620 b.  Response In PEP  
 2621 c.  Other, please describe: \_\_\_\_\_

- 2622 12. Which Policy Decision Point do you use?
- 2623 a.  Internal or built in  
 2624 b.  External XACML PDP  
 2625 c.  Other: \_\_\_\_\_

- 2626 13. Which obligations or policy languages do you use or support? (tick all that apply)
- 2627 a.  SOL1  
 2628 b.  Permis  
 2629 c.  XACML2  
 2630 d.  Other, please specify: \_\_\_\_\_  
 2631

2632 **8.5.3 Attribute Authority Questions**

2633 These questions are in addition to the WSP questions of the previous section. You should answer these  
 2634 questions if you are authority for, store, or broker user data, such as Personally Identifiable Information  
 2635 (PII).

- 2636 1. What is the nature and sensitivity of the user data you handle?
- 2637 2. What obligations do you pledge to honour with respect to user data trusted in your possession?  
 2638 *Either describe in prose or provide specific policies using Simple Obligations Language 1 (SOL1) or*  
 2639 *other obligations language you plan to use.*
- 2640 3. What obligations do you require other party to honour with respect to user data you release?  
 2641 *Either describe in prose or provide specific policies using Simple Obligations Language 1 (SOL1) or*  
 2642 *other obligations language you plan to use.*
- 2643 4. Do you have automatic mechanisms for satisfying the obligations you pledged? Please describe: \_\_\_\_\_
- 2644 5. Do you have automatic mechanisms for verifying that the requesting party pledges to respect the obli-  
 2645 gations you issue?
- 2646 6. What mechanisms do you provide to user and trust network operator to verify that you have complied  
 2647 with your pledges?
- 2648 7. What mechanisms do you have or require from others to verify that they have complied with their  
 2649 pledges?
- 2650 8. How do you protect the confidentiality of the stored user data? Describe any filesystem and crypto-  
 2651 graphic protections you employ.
- 2652 9. How do you provide Right of Access, Rectification, and Deletion?

- 2653 a.  Stand alone web GUI. URL: \_\_\_\_\_  
 2654 b.  iFrame widget Web GUI. URL: \_\_\_\_\_  
 2655 c.  Other method: \_\_\_\_\_

2656 10. In the eventuality of Rectification or Deletion, are you able to notify the parties to whom you have  
 2657 released the data in past?

2658 11. What is your policy towards data requestors who refuse to subscribe to notifications? What about  
 2659 recipients that subscribed, but refuse the actual notification?  
 2660

2661 **8.5.4 Web Service Client (WSC) Questions**

2662 A FE or WSP may act in secondary role of Web Service Client (WSC). If you call other web services  
 2663 you should answer these questions.

- 2664 1. Is your software TAS<sup>3</sup> or ID-WSF 2.0 compliant?  
 2665 Is it certified? When, by whom: \_\_\_\_\_  
 2666 2. Are you able to use Credentials and Privacy Negotiation agent?  
 2667 3. Are you able to handle Interaction Redirect if requested by WSP?  
 2668 4. What security mechanisms are you willing and able to support

- 2669 a.  Bearer Token  
 2670 b.  Holder of Key Token  
 2671 c.  X509 signature without token  
 2672 d.  None

2673 5. Which Policy Enforcement Points do you implement?

- 2674 a.  Request Out PEP  
 2675 b.  Response In PEP  
 2676 c.  Other, please describe: \_\_\_\_\_

2677 6. Which Policy Decision Point do you use?

- 2678 a.  Internal or built in  
 2679 b.  External XACML PDP  
 2680 c.  Other: \_\_\_\_\_

2681 7. Which obligations or policy languages do you use or support? (tick all that apply)

- 2682 a.  SOL1  
 2683 b.  Permis  
 2684 c.  XACML2  
 2685 d.  Other, please specify: \_\_\_\_\_

2686 8. What obligations do you pledge to honour with respect to user data returned to you?

2687 *Either describe in prose or provide specific policies using Simple Obligations Language 1 (SOL1) or*  
 2688 *other obligations language you plan to use.*

- 2689 9. What obligations do you require other party to honour with respect to user data you send?  
 2690 *Either describe in prose or provide specific policies using Simple Obligations Language 1 (SOLI) or*  
 2691 *other obligations language you plan to use.*
- 2692 10. Do you have automatic mechanisms for satisfying the obligations you pledged? Please describe: \_\_\_\_\_
- 2693 11. What mechanisms do you provide to user and trust network operator to verify that you have complied  
 2694 with your pledges?
- 2695 12. What mechanisms do you have or require from others to verify that they have complied with their  
 2696 pledges?  
 2697

2698 **8.6 Single Sign-On Identity Provider (IdP), Discovery Service, Dis-**  
 2699 **covery Registry, Identity Mapper, or Delegation Service Questions**

- 2700 1. Is your software SAML 2.0 and TAS<sup>3</sup> or ID-WSF 2.0 compliant?  
 2701 Is it certified? When, by whom: \_\_\_\_\_
- 2702 2. If your IdP or Discovery Service provides attributes, also answer questions in the Attribute Authority  
 2703 section, above.  
 2704

2705 **8.6.1 Identity Provider Questions**

- 2706 1. What authentication methods do you support (tick all that apply)
- 2707 a.  One Time Password Token, such as Yubikey, RSA token, or similar  
 2708 b.  Client certificate at user level or eID card  
 2709 c.  Mobile phone based authentication  
 2710 d.  Desktop Login based authentication  
 2711 e.  Username and password  
 2712 f.  Other, please specify: \_\_\_\_\_
- 2713 2. What user intake or vetting procedures do you have?
- 2714 3. What authentication context classes do you support and how do they map to the intake and authen-  
 2715 tication methods you support? Please specify the URIs that will be used to indicate these in various  
 2716 protocol transactions.
- 2717 4. What types of NameIDs are you willing and able to support (tick all that apply)?
- 2718 a.  Persistent per entity pseudonyms  
 2719 b.  Transient per entity  
 2720 c.  Persistent shared unique id (e.g. globally unique id or "national id")  
 2721 d.  Transient shared (e.g. random ID shared across many entities)
- 2722 5. Can you push attributes (if you can, you are also an Attribute Authority, see above)?
- 2723 6. Do you support ID-WSF 2.0 discovery bootstrap attribute?

2724

2725 **8.6.2 Discovery Service Questions**

2726 1. What registration mechanisms do you provide for WSPs?

2727 URL of the registration interface: \_\_\_\_\_

2728 2. What security mechanisms are you willing and able to support

2729 a.  Bearer Token

2730 b.  Holder of Key Token

2731 c.  X509 signature without token

2732 d.  None

2733 3. What types of NameIDs are you willing and able to support (tick all that apply)?

2734 a.  Persistent per entity pseudonyms

2735 b.  Transient per entity

2736 c.  Persistent shared unique id (e.g. globally unique id or "national id")

2737 d.  Transient shared (e.g. random ID shared across many entities)

2738 4. Can you push attributes? (if you can you are also an Attribute Authority)

2739 5. Do you support pruning discovery results by trust scoring?

2740 6. Do you support pruning discovery results based on Credentials and Privacy Negotiation?

2741

2742 **8.7 Any Other Architectural Role**

2743 As other TAS<sup>3</sup> architectural roles are less common and require special considerations, this questionnaire  
 2744 does not try to cover them. Please contact TAS<sup>3</sup> consortium for further guidance.

## Bibliography

- [AAPML] Prateek Mishra, ed.: "AAPML: Attribute Authority Policy Markup Language", Working Draft 08, Nov. 28, 2006, Liberty Alliance / Oracle. <http://www.oracle.com/technology/tech/standards/idm/igf/pdf/IGF-AAPML-spec-08.pdf>
- [AcctSvc] "Liberty ID-WSF Accounting Service Specification"
- [AdvClient] "Liberty ID-WSF Advanced Client Technologies Overview", liberty-idwsf-adv-client-v1.0.pdf
- [AeGArch07] "D3.1 Access-eGov Platform Architecture", Access-eGov consortium, Feb 12, 2007. [http://www.accessegov.org/acegov/uploadedFiles/webfiles/cffile\\_4\\_3\\_07\\_3\\_25\\_17\\_PM.pdf](http://www.accessegov.org/acegov/uploadedFiles/webfiles/cffile_4_3_07_3_25_17_PM.pdf), also <http://www.accessegov.org/acegov/web/uk/index.jsp?id=50268>
- [Alberts01] Alberts, C. J., & Dorofee, A. J. (2001). OCTAVE Criteria Version 2.0. Tech. report CMU/SEI-2001-TR-016. ESC-TR-2001-016.
- [AMQP06] "AMQP: A General-Purpose Middleware Standard" (a.k.a Advanced Message Queuing Protocol), 2006.
- [Anderson07] Anne Anderson: "Web Services Profile of XACML (WS-XACML) Version 1.0", Working Draft 10, OASIS XACML Technical Committee, 10 August 2007, available at <http://www.oasis-open.org/committees/download.php/24950/xacml-3.0-profile-webservices-v1-wd-10.zip>
- [BraberEA07] Den Braber, F., Hogganvik, I., Lund, M. S., Stølen, K., & Vraalsen, F. (2007). Model-based security analysis in seven steps - a guided tour to the CORAS method. *BT Technology Journal*, 25(1), pp. 101-117.
- [CardSpace] InfoCard protocol (aka CardSpace) from Microsoft
- [CARML] Phil Hunt and Prateek Mishra, eds.: "Liberty IGF Client Attribute Requirements Markup Language (CARML) Specification", Draft 1.0-12, Liberty Alliance, 2008. [http://www.projectliberty.org/liberty/resource\\_center/specifications/igf\\_1\\_0\\_specs](http://www.projectliberty.org/liberty/resource_center/specifications/igf_1_0_specs)
- [Castano07] Castano, S., Ferrara, A., Montanelli, S., Hess, G. N., and Bruno, S. (2007). State of the art on ontology coordination and matching. Report FP6-027538, BOEMIE.
- [Chadwick08] David Chadwick: "Functional Components of Grid Service Provider Authorisation Service Middleware", Open Grid Forum, 17 September, 2008. (\*\*\* AuthzFunc0.7.doc)
- [Chadwick09] David Chadwick: "FileSpace - An Alternative to CardSpace that supports Multiple Token Authorisation and Portability Between Device". Presented at IDtrust 2009, the 8th Symposium on Identity and Trust on the Internet, NIST, Gaithersberg, April 2009. Available from <http://middleware.internet2.edu/idtrust/2009/papers/08-chadwick-filespace.pdf>
- [ChadwickEA09] David W Chadwick, Sassa Otenko and Tuan Anh Nguyen. "Adding Support to XACML for Multi-Domain User to User Dynamic Delegation of Authority". *International Journal of Information Security*. Volume 8, Number 2 / April, 2009 pp 137-152. DOI 10.1007/s10207-008-0073-y
- [ChadwickEA09b] David W Chadwick, Linying Su, Romain Laborde: "Use of XACML Request Context to Obtain an Authorisation Decision". GFD.159. 13 November 2009. Available from <http://www.ogf.org/documents/GFD.159.pdf>

- [ChadwickSu09] David Chadwick, Linying Su: "Use of WS-TRUST and SAML to access a Credential Validation Service". GFD.157. 13 November 2009. Available from <http://www.ogf.org/documents/GFD.157.pdf>
- [CogWalkthruWeb] <http://www.cc.gatech.edu/classes/cs3302/documents/cog.walk.html>
- [CVS-SAML-WS-Trust] David Chadwick and Linying Su: "Use of WS-TRUST and SAML to access a Credential Validation Service", Open Grid Forum, 2008. (\*\*\*) WS-TrustProfile0.8.doc)
- [DahlEA07] Dahl, H., Hogganvik, I., & Stølen, K. (2007). Structured semantics for the CORAS security risk modelling language. Pre-proceedings of the 2nd International Workshop on Interoperability Solutions on Trust, Security, Policies and QoS for Enhanced Enterprise Systems (IS-TSPQ'07), (pp. 79-92).
- [DesignPat] "Liberty ID-WSF Design Patterns", liberty-idwsf-dp-v1.0.pdf
- [Dieng98] Dieng, R. and Hug, S. (1998). Comparison of "personal ontologies" represented through conceptual graphs. In Proceedings of the 13th European Conference on Artificial Intelligence (ECAI 98), pages 341-345, Brighton, UK.
- [Disco2] Cahill, ed.: "Liberty ID-WSF Discovery service 2.0", liberty-idwsf-disco-svc-2.0-errata-v1.0.pdf from [http://projectliberty.org/resource\\_center/](http://projectliberty.org/resource_center/)
- [Disco12] Liberty ID-WSF Discovery service 1.2 (liberty-idwsf-disco-svc-v1.2.pdf)
- [DST11] Liberty DST v1.1
- [DST21] Sampo Kellomäki and Jukka Kainulainen, eds.: "Liberty Data Services Template 2.1", Liberty Alliance, 2007. liberty-idwsf-dst-v2.1.pdf from [http://projectliberty.org/resource\\_center/specifications/](http://projectliberty.org/resource_center/specifications/)
- [DST20] Sampo Kellomäki and Jukka Kainulainen, eds.: "Liberty DST v2.0", Liberty Alliance, 2006.
- [Enisa10] Inventory of Risk Management / Risk Assessment Methods. [http://rm-inv.enisa.europa.eu/rm\\_ra\\_methods.html](http://rm-inv.enisa.europa.eu/rm_ra_methods.html) (fetched 25.6.2010)
- [FF12] Liberty ID Federation Framework 1.2, Protocols and Schemas
- [FMC03] Frank Keller, Siegfried Wendt: "FMC: An Approach Towards Architecture-Centric System Development", Hasso Plattner Institute for Software Systems Engineering, 2003.
- [FMCWeb] "Fundamental Modeling Concepts" <http://fmc-modeling.org/>
- [GiraoSarma10] João Girão and Amardeo Sarma: "IDentity Engineered Architecture (IDEA)", in Towards the Future Internet, G. Tselentis et al. (Eds.), IOS Press, 2010. (STAL9781607505396-0085.pdf)
- [HafnerBreu09] Hafner & Breu: "Security Engineering for Service-Oriented Architectures", Springer, 2009.
- [Hardt09] Dick Hardt and Yaron Goland: "Simple Web Token (SWT)", Version 0.9.5.1, Microsoft, Nov. 4, 2009 (SWT-v0.9.5.1.pdf)
- [IAF] Russ Cutler, ed.: "Identity Assurance Framework", Liberty Alliance, 2007. File: liberty-identity-assurance-framework-v1.0.pdf (from [http://projectliberty.org/liberty/resource\\_center/papers](http://projectliberty.org/liberty/resource_center/papers))

- [ICAMSAML2] Terry McBride and Dave Silver, eds.: "Federal Identity, Credentialing, and Access Management Security Assertions Markup Language (SAML) 2.0 Profile", version 0.1.0 draft, Feb 17, 2010, Federal-ICAMSC-SAML-20-Profile-Draftv010-36529.pdf
- [IDDAP] Sampo Kellomäki, ed.: "Liberty Identity based Directory Access Protocol", Liberty Alliance, 2007.
- [IDFF12] <http://www.projectliberty.org/resources/specifications.php>
- [IDFF12meta] Peted Davis, ed., "Liberty Metadata Description and Discovery Specification", version 1.1, Liberty Alliance Project, 2004. (liberty-metadata-v1.1.pdf)
- [IDPP] Sampo Kellomäki, ed.: "Liberty Personal Profile specification", Liberty Alliance, 2003.
- [IDWSF08] Conor Cahill et al.: "Liberty Alliance Web Services Framework: A Technical Overview", Liberty Alliance, 2008. File: idwsf-intro-v1.0.pdf (from [http://projectliberty.org/liberty/resource\\_center/papers](http://projectliberty.org/liberty/resource_center/papers))
- [IDWSF2IOP] Eric Tiffany, ed.: "Liberty ID-WSF 2.0 Interoperability Testing Procedures", Version Draft 1.0-01, 16. Aug. 2006. File: ID-WSF-2-0-TestProcedures-v1-01.pdf, from <http://projectliberty.org/>
- [IDWSF2MRD] "Liberty ID-WSF 2.0 Marketing Requirements Document", Liberty Alliance, 2006. File: liberty-idwsf-2.0-mrd-v1.0.pdf (from [http://projectliberty.org/liberty/strategic\\_initiatives/requirements/](http://projectliberty.org/liberty/strategic_initiatives/requirements/))
- [IDWSF2Overview] "Liberty ID-WSF Architecture Overview", liberty-idwsf-overview-v2.0.pdf from [http://projectliberty.org/resource\\_center/specifications](http://projectliberty.org/resource_center/specifications)
- [IDWSF2SCR] "Liberty ID-WSF 2.0 Static Conformance Requirements", liberty-idwsf-2.0-scr-1.0-errata-v1.0.pdf
- [IDWSFSecPriv] "Liberty ID-WSF Security & Privacy Overview", liberty-idwsf-security-privacy-overview-v1.0.pdf from [http://projectliberty.org/resource\\_center/specifications/](http://projectliberty.org/resource_center/specifications/)
- [IGF] "An Overview of the Identity Governance Framework", Liberty Alliance, 2007. File: overview-id-governance-framework-v1.0.pdf (from [http://projectliberty.org/liberty/resource\\_center/papers](http://projectliberty.org/liberty/resource_center/papers))
- [Interact2] "Liberty ID-WSF Interaction Service", liberty-idwsf-interaction-svc-2.0-errata-v1.0.pdf from [http://projectliberty.org/resource\\_center/specifications/](http://projectliberty.org/resource_center/specifications/)
- [ISO27001] ISO standard 27001: <http://www.iso.org>
- [Kellomaki08] Sampo Kellomäki: "Query Extension for SAML AuthnRequest", feature request to OASIS Security Services Technical Committee (SSTC), 2008. See OASIS SSTC mailing list archive.
- [Levenshtein66] Levenshtein, V. I. (1966). Binary codes capable of correcting deletions, insertions and reversals. Soviet Physics Doklady, 10:707+.
- [LibertyInterFed] Carolina Canales Valenzuela, Sampo Kellomäki, eds.: "Access to Identity-Enabled Web Services in Cross-Border, Inter-Federation Scenarios", Liberty Alliance, 2007. File: access-to-identity-enabled-services-in-inter-cot-scenarios-v1.0.pdf (from [http://projectliberty.org/liberty/resource\\_center/papers](http://projectliberty.org/liberty/resource_center/papers))
- [LibertyLegal] Victoria Sheckler, ed.: "Contractual Framework Outline for Circles of Trust", Liberty Alliance, 2007. File: Liberty Legal Frameworks.pdf (from [http://projectliberty.org/liberty/resource\\_center/papers](http://projectliberty.org/liberty/resource_center/papers))

- [LibertyXF] Sampo Kellomäki, ed.: "Cross Operation of Single Sign-On, Federation, and Identity Web Services Frameworks", Liberty Alliance, 2006.
- [Madsen03] Paul Madsen: "WS-Trust: Interoperable Security for Web Services" Available from <http://www.xml.com/pub/a/ws/2003/06/24/ws-trust.html>
- [Mbanaso09] U.M. Mbanaso, G.S. Cooper, David Chadwick, Anne Anderson: "Obligations of Trust for Privacy and Confidentiality in Distributed Transactions", Internet Research. Vol 19 No 2, 2009, pp. 153-173.
- [Meier08] J.D. Meier: "Threats, Attacks, Vulnerabilities, and Countermeasures", 30.3.2008. <http://shapingsoftware.com/2008/03/30/threats-attacks-vulnerabilities-and-countermeasures/>
- [Meier09] J.D. Meier: "Security Hot Spots", 9.3.2009. <http://shapingsoftware.com/2009/03/09/security-hot-spots/>
- [Microsoft06] Microsoft Centre of Excellence. (2006). The Security Risk Management Guideline. Microsoft Solutions for Security and Compliance.
- [MS-MWBF] Microsoft Web Browser Federated Sign-On Protocol Specification, 20080207, <http://msdn2.microsoft.com/en-us/library/cc236471.aspx>
- [Nagios] "System, Network, and Application Monitor", the latest incarnation of the Satan and Net Saint saga, <http://www.nagios.org/>
- [NexofRA09] "Deliverable D6.2 RA Model V2.0", All NEXOF-RA Partners, NESSI Strategic Project and External Contributors, 2009.
- [NIST-SP800-30] Gary Stoneburner, Alice Goguen, and Alexis Feringa: "Risk Management Guide for Information Technology Systems", Recommendations of the National Institute of Standards and Technology, NIST, 2002. <http://csrc.nist.gov/publications/nistpubs/800-30/sp800-30.pdf>
- [NIST-SP800-42] John Wack, Miles Tracy and Murugiah Souppaya: "Guideline Network Security", Recommendations of the National Institute of Standards and Technology, NIST, 2002. <http://csrc.nist.gov/publications/nistpubs/800-30-42/sp800-42.pdf>
- [NIST-SP800-63] William E. Burr, Donna F. Dodson, Ray A. Perlner, W. Timothy Polk, Sarbari Gupta, Emad A. Nabbus: "Electronic Authentication Guideline", Recommendations of the National Institute of Standards and Technology, NIST Special Publication 800-63-1, Feb 2008. <http://csrc.nist.gov/publications/nistpubs/>
- [OAUTH] <http://oauth.net/>
- [OpenID] <http://openid.net/>
- [OWL-S-Web] David Martin, ed.: "OWL-S: Semantic Markup for Web Services", W3C, 22. Nov, 2004. <http://www.w3.org/Submission/OWL-S/>
- [PCI08] "Payment Card Industry Data Security Standard", Version 1.2, Oct 2008, PCI Security Standards Council. Document [pci\\_dss\\_v1-2.pdf](https://www.pcisecuritystandards.org/security_standards/pci_dss.shtml) from [https://www.pcisecuritystandards.org/security\\_standards/pci\\_dss.shtml](https://www.pcisecuritystandards.org/security_standards/pci_dss.shtml)
- [Peeters09] Roel Peeters, Koen Simoens, Danny De Cock, and Bart Preneel: "Cross-Context Delegation through Identity Federation", KUL 2009 (To be published?)
- [PeopleSvc] "Liberty ID-WSF People Service Specification", [liberty-idwsf-people-service-1.0-errata-v1.0.pdf](http://projectliberty.org/resource_center/specifications/liberty-idwsf-people-service-1.0-errata-v1.0.pdf) from [http://projectliberty.org/resource\\_center/specifications/](http://projectliberty.org/resource_center/specifications/)

- [PERMIS] D.W.Chadwick and A. Otenko: "The PERMIS X.509 Role Based Privilege Management Infrastructure". Future Generation Computer Systems, Vol 19, Issue 2, Feb 2003. pp 277-289
- [RFC1157] J. Case et al.: " A Simple Network Management Protocol (SNMP)", RFC 1157, 1990.
- [RFC1950] P. Deutch, J-L. Gailly: "ZLIB Compressed Data Format Specification version 3.3", Aladdin Enterprises, Info-ZIP, May 1996
- [RFC1951] P. Deutch: "DEFLATE Compressed Data Format Specification version 1.3", Aladdin Enterprises, May 1996
- [RFC1952] P. Deutch: "GZIP file format specification version 4.3", Aladdin Enterprises, May 1996
- [RFC2119] S. Bradner, ed.: "Key words for use in RFCs to Indicate Requirement Levels", Harvard University, 1997.
- [RFC2138] C. Rigney et al.: "Remote Authentication Dial In User Service (RADIUS)", RFC 2138, April 1997.
- [RFC2139] C. Rigney: "RADIUS Accounting", RFC 2139, April 1997.
- [RFC2246] T. Dierks and C. Allen: "The TLS Protocol Version 1.0", RFC 2246, January 1999.
- [RFC2251] M. Wahl, T. Howes, S. Kille: "Lightweight Directory Access Protocol (v3)", RFC 2251, December 1997.
- [RFC2256] Wahl, M., "A Summary of the X.500(96) User Schema for use with LDAPv3", RFC 2256, December 1997.
- [RFC2560] Myers et al., "X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP", RFC 2560, June 1999.
- [RFC2798] M. Smith: "Definition of the inetOrgPerson LDAP Object Class", Netscape Communications, RFC 2798, April 2000.
- [RFC3548] S. Josefsson, ed.: "The Base16, Base32, and Base64 Data Encodings", July 2003. (Section 4 describes Safebase64)
- [RFC3588] P. Calhoun et al.: "Diameter Base Protocol", RFC 3588, September 2003.
- [RFC3768] R. Hinden, ed.: "Virtual Router Redundancy Protocol (VRRP)", RFC 3768, April 2004.
- [SAML2LOA] OASIS. "Level of Assurance Authentication Context Profiles for SAML 2.0" Working Draft 01. 01 July 2008
- [SAML11core] SAML 1.1 Core, OASIS, 2003
- [SAML11bind] "Bindings and Profiles for the OASIS Security Assertion Markup Language (SAML) V1.1", Oasis Standard, 2.9.2003, oasis-sstc-saml-bindings-1.1
- [SAML2core] "Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML) V2.0", Oasis Standard, 15.3.2005, saml-core-2.0-os
- [SAML2prof] "Profiles for the OASIS Security Assertion Markup Language (SAML) V2.0", Oasis Standard, 15.3.2005, saml-profiles-2.0-os
- [SAML2profErrata] OASIS. "Profiles for the OASIS Security Assertion Markup Language (SAML) V2.0 - Errata Composite Working Draft", 12 February 2006

- [SAML2bind] "Bindings for the OASIS Security Assertion Markup Language (SAML) V2.0", Oasis Standard, 15.3.2005, saml-bindings-2.0-os
- [SAML2context] "Authentication Context for the OASIS Security Assertion Markup Language (SAML) V2.0", Oasis Standard, 15.3.2005, saml-authn-context-2.0-os
- [SAML2meta] Cantor, Moreh, Philpott, Maler, eds., "Metadata for the OASIS Security Assertion Markup Language (SAML) V2.0", Oasis Standard, 15.3.2005, saml-metadata-2.0-os
- [SAML2security] "Security and Privacy Considerations for the OASIS Security Assertion Markup Language (SAML) V2.0", Oasis Standard, 15.3.2005, saml-sec-consider-2.0-os
- [SAML2conf] "Conformance Requirements for the OASIS Security Assertion Markup Language (SAML) V2.0", Oasis Standard, 15.3.2005, saml-conformance-2.0-os
- [SAML2glossary] "Glossary for the OASIS Security Assertion Markup Language (SAML) V2.0", Oasis Standard, 15.3.2005, saml-glossary-2.0-os
- [SAML2SimpleSign] "SAML 2.0 POST Simple Sign Binding", OASIS, 2008.
- [Schema1-2] Henry S. Thompson et al. (eds): XML Schema Part 1: Structures, 2nd Ed., WSC Recommendation, 28. Oct. 2004, <http://www.w3.org/2002/XMLSchema>
- [SecMech2] "Liberty ID-WSF 2.0 Security Mechanisms", liberty-idwsf-security-mechanisms-core-2.0-errata-v1.0.pdf from [http://projectliberty.org/resource\\_center/specifications](http://projectliberty.org/resource_center/specifications)
- [Shibboleth] <http://shibboleth.internet2.edu/shibboleth-documents.html>
- [SHPS] Conor Cahill, et al.: "Service Hosting and Proxying Service Specification", Liberty Alliance Project, 15. Dec. 2006.
- [Siemens10] Cram Methods <http://www.cramm.com> (fetched in 25.6.2010)
- [SOAPAuthn2] "Liberty ID-WSF Authentication, Single Sign-On, and Identity Mapping Services Specification", liberty-idwsf-authn-svc-2.0-errata-v1.0.pdf from [http://projectliberty.org/resource\\_center/specifications/](http://projectliberty.org/resource_center/specifications/)
- [SOAPBinding2] "Liberty ID-WSF SOAP Binding Specification", liberty-idwsf-soap-binding-2.0-errata-v1.0.pdf from [http://projectliberty.org/resource\\_center/specifications](http://projectliberty.org/resource_center/specifications)
- [SOX02] "Sarbanes-Oxley Act of 2002", Public Law 107-204, United States, 2002. [http://frwebgate.access.gpo.gov/cgi-bin/getdoc.cgi?dbname=107\\_cong\\_public\\_laws&docid=f:publ204.107](http://frwebgate.access.gpo.gov/cgi-bin/getdoc.cgi?dbname=107_cong_public_laws&docid=f:publ204.107)
- [SUBS2] "Liberty ID-WSF Subscriptions and Notifications Specification", liberty-idwsf-subsv1.0.pdf from [http://projectliberty.org/resource\\_center/specifications/](http://projectliberty.org/resource_center/specifications/)
- [SwiderskiSnyder04] Frank Swiderski and Window Snyder. Threat Modeling. Microsoft Press, 2004.
- [SWIG] Simplified Interface and Wrapper Generator by Dave Beazley. [www.swig.org](http://www.swig.org)
- [TAS3ARCH] Sampo Kellomäki, ed.: "TAS3 Architecture", TAS3 Consortium, 2009. Document: tas3-arch-vXX.pdf, also deliverable D2.1, document: tas3-deliv-2\_1-arch-v17\_2.pdf
- [TAS3BIZ] Luk Vervenne, ed.: "TAS3 Business Model", TAS3 Consortium, 2009.
- [TAS3COMPLIANCE] Sampo Kellomäki, ed.: "TAS3 Compliance Requirements", TAS3 Consortium, 2009. Document: tas3-compliance-vXX.pdf
- [TAS3CONSOAGMT] "TAS3 Consortium Agreement", TAS3 Consortium, 2008. (Not publicly available.)

- [TAS3D12DESIGNRAR] David Chadwick (Kent), Seda Gürses (KUL), eds.: "Requirements Assessment Report", TAS3 Consortium, 20090102. Document: TAS3\_D1p2\_Requirements\_Assesment\_Report\_1\_V1p0.pdf
- [TAS3D14DESIGNREQ] Gilles Montagnon (SAP), ed.: "Design Requirements", TAS3 Consortium, 20081221. Document: TAS3\_D1p4\_Design\_Requirements\_1\_V2p0.pdf
- [TAS3D22UPONTO] Quentin Reul (VUB), ed.: "Common Upper Ontologies", TAS3 Consortium, Deliverable D2.2, 7.5.2009. Document: D2.2\_ver1.7.pdf
- [TAS3D41ID] Sampo Kellomäki, ed.: "Identifier and Discovery Function", TAS3 Deliverable 4.1, 2009. Document: tas3-disco-v01.pdf
- [TAS3D42Repo] David Chadwick, ed.: "Specification of information containers and authentic repositories", TAS3 Deliverable 4.2, 2009.
- [TAS3D62Contract] Joseph Alhadeff, Brendan Van Alsenoy: "Contractual Framework", v3.0, TAS3 Deliverable D6.1, December 2009.
- [TAS3D71IdMANAz] TAS3 Deliverable 7.1. "Design of Identity Management, Authentication and Authorization Infrastructure" 3 Jan 2009.
- [TAS3D81RepoSW] "Software Documentation System: Repository Services", UniKOLD, TAS3 Deliverable 8.1, 2009.
- [TAS3D82BackOffice] "Back Office Services with Documentation", TAS3 Consortium, 2009.
- [TAS3D83CliSW] "TAS3 Client Software with User Guide", TAS3 Consortium, 2009.
- [TAS3D91PilotUC] "Pilot Use Cases", Deliverable D9.1, TAS3 Consortium, 2009.
- [TAS3DOW] "TAS3 Description of Work", TAS3 Consortium, 2008. (Not publicly available.) File: TAS3\_DescriptionOfWork.DoW.technical.annex.final.version.20071030.pdf
- [TAS3GLOS] Quentin Reul (VUB), ed.: "TAS3 Glossary", TAS3 Consortium, 2009. Document: tas3-glossary-vXX.pdf
- [TAS3PROTO] Sampo Kellomäki, ed.: "TAS3 Protocols and Concrete Architecture", TAS3 Consortium, 2009. Document: tas3-proto-vXX.pdf
- [TAS3RISK] Magalie Seguran, ed.: "TAS3 Risk Assessment", TAS3 Consortium, 2010. Document: tas3-risk-vXX.pdf
- [TAS3TECHQUIZ] Sampo Kellomäki, ed.: "TAS3 Technical Self-Assessment Questionnaire", TAS3 Consortium, 2010. Document: tas3-tech-quiz-vXX.pdf
- [TAS3THREAT] Sampo Kellomäki, ed.: "TAS3 Threat Analysis", TAS3 Consortium, 2009. Document: tas3-threats-vXX.pdf
- [TAS3USERCENT] Gilles Montagnon, ed.: "TAS3 User Centricity Report", TAS3 Consortium, 2010. Document: tas3-user-cent-vXX.pdf
- [TAS3WP] "TAS3 Architecture White Paper", TAS3 Consortium, 2009 (as of 20090324 to be published).
- [Tom09] Allen Tom, et al.: "OAuth Web Resource Authorization Profiles (OAuth WRAP)", Version 0.9.7.2, Google, Microsoft, and Yahoo, Nov. 5, 2009 (WRAP-v0.9.7.2.pdf)
- [TrustBuilder2] Adam J. Lee, Marianne Winslett and Kenneth J. Perano: "TrustBuilder2: A Reconfigurable Framework for Trust Negotiation", IFIP Trust Management Conference, June 2009.

- [UML2] [http://www.sparxsystems.com.au/resources/uml2\\_tutorial/](http://www.sparxsystems.com.au/resources/uml2_tutorial/)
- [UNDP07] "e-Government Interoperability Guide", United Nations Development Programme, 2007. <http://www.apdip.net/projects/gif/GIF-Guide.pdf>
- [VenturiEA08] V. Venturi, et al.: "Use of SAML to retrieve Authorization Credentials", Open Grid Forum, 2008. (\*\*\*) Attribute PullProfilev1.5.doc; CVS related)
- [Wharton94] C. Wharton et al. "The cognitive walkthrough method: a practitioner's guide" in J. Nielsen & R. Mack "Usability Inspection Methods" pp. 105-140, Wiley, 1994.
- [WSML-Web] "Web Services Modelling Language" <http://www.wsmo.org/wsm/>
- [WSMO05] D. Roman, U. Keller, H. Lausen, J. de Bruijn, R. Lara, M. Stollberg, A. Polleres, C. Feier, C. Bussler, and D. Fensel (2005). "Web Service Modeling Ontology". In Applied Ontology 1, pages 77-106.
- [WSMO-Web] "Web Services Modelling Ontology" <http://www.wsmo.org/>
- [WSPolicy] Bajaj et al.: "Web Services Policy Framework (WS-Policy) and Web Services Policy Attachment (WS-PolicyAttachment)", W3C, March 2006. <http://schemas.xmlsoap.org/ws/2004/09/policy/>
- [WSTrust] "WS-Trust 1.3", CD 6, OASIS, Sept 2006. (\*\*\*) WS-Trust, STS, etc.)
- [X520] ITU-T Rec. X.520, "The Directory: Selected Attribute Types", 1996.
- [X521] ITU-T Rec. X.521, "The Directory: Selected Object Classes", 1996.
- [XACML2] "eXtensible Access Control Markup Language (XACML)" v2.0, OASIS Standard, February 2005. From [http://www.oasis-open.org/committees/tc\\_home.php?wg\\_abbrev=xacml](http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xacml)
- [XACML2SAMLold] "SAML 2.0 Profile of XACML, Version 2, Working Draft 5", 19 July 2007, OASIS. (\*\*\*) instead of "SAML 2.0 profile of XACML v2.0, ERRATA, Working Draft 01, 17 November 2005" which is the version that the profile is currently based on; XACML-ContextProfile1.1.doc from Open Grid Forum - OGF)
- [XACML2SAML] "SAML 2.0 Profile of XACML, Version 2, Committee Draft", 16 April 2009
- [XML] <http://www.w3.org/TR/REC-xml>
- [XML-C14N] XML Canonicalization (non-exclusive), <http://www.w3.org/TR/2001/REC-xml-c14n-20010315>; J. Boyer: "Canonical XML Version 1.0", W3C Recommendation, 15.3.2001, <http://www.w3.org/TR/xml-c14n>, RFC3076
- [XML-EXC-C14N] Exclusive XML Canonicalization, <http://www.w3.org/TR/xml-exc-c14n/>
- [XMLDSIG] "XML-Signature Syntax and Processing", W3C Recommendation, 12.2.2002, <http://www.w3.org/TR/xmlsig-core>, RFC3275
- [XMLENC] "XML Encryption Syntax and Processing", W3C Recommendation, 10.12.2002, <http://www.w3.org/TR/xmlenc-core>
- [XPATH99] James Clark and Steve DeRose, eds. "XML Path Language (XPath) Version 1.0", W3C Recommendation 16 November 1999. From: <http://www.w3.org/TR/xpath>
- [ZXIDREADME] Sampo Kellomäki: "README.zxid" file from [zxid.org](http://zxid.org), 2009.