SEVENTH FRAMEWORK PROGRAMME
Challenge 1
Information and Communication Technologies

Trusted Architecture for Securely Shared Services

Document Type:	Deliverable
Title:	**TAS^3 Protocols, API, and Concrete Architecture**
Work Package:	WP2
Deliverable Nr:	D2.4
Dissemination:	Public
Preparation Date:	30 June 2010
Version:	14 (1.67)

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.

SEVENTH FRAMEWORK PROGRAMME

The TAS^3 Consortium

	Beneficiary Name	Country	Short	Role
1	K.U. Leuven	BE	KUL	Coordinator
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	S&T Coordn
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
18	Medisoft	NL	MEDI	Partner
19	Symlabs	PT	SYM	Partner

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

Contributors

	Name	Organisation
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

List of Figures

Keyword List

1 Architecture, Protocol, Implementation, API, Security, Trust, Privacy

Protocols and Concrete Architecture Executive Summary

This document specifies a set of protocol level interoperability profiles, usually leveraging open standards, deployment scenarios, APIs, and other considerations that constitute the official way to deploy version 1 of TAS^3 architecture, see [**?**]. The purpose of defining these specifics is to enable multiple independent implementations of TAS^3 to be wire protocol interoperable (and to limited extent also API interoperable). TAS^3 reference implementation and reference deployment will behave essentially as described in this document.

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

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

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

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

We provide guidance regarding possible integration and deployment scenarios and illustrate how TAS^3 Architecture can be deployed in a resilient and redundant way.

Neither this document nor the TAS^3 Architecture [**?**] mandate use of a particular deployment or software architecture (although the integration scenarios suggest a recommended one), implementers are free to organize their software and deployment in other ways as long as the wire protocol compatibility is maintained and all signature generation and validation steps, as well as trust determinations, and authorizations are implemented.

The Annex gives some example protocol messages.

1 Introduction

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

This document uses the keywords (e.g. MUST, SHOULD) of [**?**]. 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 [**?**].

1.1 Standardized Wire Protocol Interfaces

TAS^3 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^3 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 2.12, 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^3 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.

1.2 Composition and Co-location of Architectural Components

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

When implementing practical systems, it often turns out that many of the architecturally designed boxes are in fact implementable by one software module. For example, with reference to Fig-2.3 of [**?**], it is clear that a software module called "Service Requester" may exist, realizing Rq-PEP-Out, Rq-PEP-In, and Stack components all together without them being necessarily separable. Such composition does not harm interoperability as those submodules of Service Requester were always meant to be part of the same process and to communicate via function call interfaces. Indeed, the official TAS^3 API, see section 3, lumps all these in one function call: *tas3_call()*. However all external interfaces from *tas3_call()*, such as authorization, discovery, and web service call, do speak standard protocols as profiled in this document.

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

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

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

- anything vs. Audit
- SP vs. IdP (some exceptions apply)
- SP vs. ID Mapping and Discovery
- SP vs. Delegation
- IdP vs. Authorization (some exceptions apply)

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

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

2 Protocols and Profiles

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

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

2.1 Signature and Encryption Considerations

1. When applying XML Encryption [**?**], e.g. in EncryptedAssertion, EncryptedID, or EncryptedAttribute, the nested method of key conveyance MUST be used, i.e. key is carried in `EncryptedAssertion/EncryptedData/Key` The sibling method that uses `EncryptedAssertion/EncryptedKey` MUST NOT be used.

2. When applying [**?**], the InclusiveNamespaces/@PrefixList MUST NOT contain prefixes that are not defined in the XML document.

2.2 Supported Authentication and Login Systems

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*.

2.2.1 System Entity Authentication

TAS^3 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.

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

2.2.2 SAML

> 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^3 compliant in future.

1. TAS^3 adopts SAML 2.0 Assertions, see [**?**], as primary and recommended token format. Alternatives such as SAML 1.1 or Simple Web Token (SWT) [**?**] 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.

2. TAS^3 adopts SAML 2.0 as primary and RECOMMENDED SSO system, see [**?**]. (Req. *D1.2-3.10-JITPerm*)

3. TAS^3 RECOMMENDS that SAML 2.0 implementations are Liberty Alliance Certified.

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

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

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

7. Redirect Single Logout Profile MUST be supported, see [**?**] and [**?**].

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

9. Other SAML profiles MAY be supported

10. SAML metadata MUST be supported, see [**?**]

11. Well Known Location (WKL) method of metadata publishing MUST be supported, see [**?**] 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.

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

12. In redirect binding [**?**] deflate compression MUST be used. [**?**] format MUST NOT be used.

2.2.2.1 Authentication Request

1. MUST use `NameIDPolicy/@Format` of Persistent ("urn:oasis:names:tc:SAML:2.0:nameid-format:persistent") when implementing Pull Model (Req. *D1.2-7.8-NoColl*).

2. MUST use `NameIDPolicy/@Format` of Transient ("urn:oasis:names:tc:SAML:2.0:nameid-format:transient") when implementing Linking Service model.

3. MUST set `NameIDPolicy/@SPNameQualifier`

4. MUST set `NameIDPolicy/@AllowCreate` flag at all times true

5. SHOULD not set `IsPassive` flag (in some cases there may be justified reasons to do otherwise)

6. MUST use `AssertionConsumerServiceIndex`

7. MUST NOT use `ProtocolBinding` or `AssertionConsumerServiceURL`

8. Step-up authentication, using Authentication Context Class References MUST be supported.

9. SHOULD use `AttributeConsumingServiceIndex` attribute, which refers to a section of the metadata, as way of selecting the attributes that are returned in the authentication response. Reader should be aware that new proposals for solving this issue more dynamically have been submitted to OASIS Security Services Technical Committee, e.g. [**?**]. It should also be noted that the returned attributes are always at discretion of the IdP.

2.2.2.2 Authentication Response

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

1. MUST contain `<sa:AuthnStatement>`

2. MUST specify the Level of Authentication as `AuthnStatement/AuthnContext/AuthnContextClassRef`.

3. MUST use the LoA profile [**?**] to return LoA to the SP.

4. SHOULD have `AudienceRestriction/Audience` element referencing the SP.

5. MAY contain `<AttributeStatement>` detailing user's attributes as relevant to SP and/or requested using AttributeConsumingServiceIndex.

6. SHOULD have an `<AttributeStatement>` containing a discovery bootstrap (attribute named "urn:liberty:disco:2006-08:DiscoveryEPR" whose value is an endpoint reference) as described in [**?**] section 4 "Discovery Service ID-WSF EPR conveyed via a Security Token".

7. MAY have additional Attribute Statements conveying other endpoint references. Rather than providing additional EPRs at SSO, using discovery is RECOMMENDED. If additional EPRs are passed, the attributes SHOULD be named "urn:liberty:disco:2006-08:DiscoveryEPR" even if they do not refer to discovery service. The SP, when seeing "urn:liberty:disco:2006-08:DiscoveryEPR" attribute MUST look at the `Attribute/AttributeValue/EndpointReference/Metadata/ServiceType` element to determine the type of the end point reference. The SP SHOULD consider any attribute whose value is an `<a:EndpointReference>` to be a bootstrap.

2.2.3 Proxy IdP Profile

To adapt non-TAS^3 IdPs to TAS^3 environment, the strategy of using SAML2 Proxy IdP profile is recommended. The TAS^3 SP redirects the use to a TAS^3 enabled *proxy IdP* (aka "middle IdP"), which then offers the user a choice of actual (non-TAS^3) IdP to use and plays the SAML SP role towards that IdP. When the user has been authenticated, the assertion is returned to the middle IdP, which will use information in it to mint an assertion that is returned to the TAS^3 enabled SP. The TAS^3 assertion SHOULD contain the attributes of the original assertion. It MAY contain the original assertion as well, if audience restriction permits this.

The Proxy IdP Profile can also used for facilitation of interoperation across trust networks. SPs in one trust network use the IdP in their home trust network, which then contacts the foreign IdP. This way only the home trust network's IdP needs to have trust relationship with the foreign IdP. This is much more scalable than each SP having to trust directly the foreing IdPs. See [**?**] for further discussion.

The Proxy IdP Profile is described [**?**] section `3.4.1.5` "Proxying" (pp.54-55) and also in [**?**] section 3.3.1 IdP Proxy Feature (pp.11-12), as well as in [**?**] Step D (p.17-19) associated with "IdP Extended" and "SP Extended" conformance modes.

2.2.4 Shibboleth

Shibboleth MAY be supported. Shibboleth based on SAML 2.0 is RECOMMENDED. Supporting Shibboleth enables higher education institutions to adopt TAS^3 with minimal reconfiguration and reinvestment.

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

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

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

2.2.5 eID and Other Smart Cards

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

2.2.6 One-Time-Password Tokens

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

2.2.7 OpenID

OpenID [?] MAY be supported. If supported, OpenID 2.0 MUST be used as earlier versions have known security flaws.

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

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

2.2.8 CardSpace / InfoCard and WS-Federation

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

2.2.9 CA / Netegrity Siteminder Proprietary SSO

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

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

2.2.10 Citrix, Sun, and other proprietary SSO

MAY be supported. However, we have not validated whether it is possible to implement TAS^3 architecture using these.

2.2.11 Web Local Login

We have not validated whether it is possible to implement TAS^3 architecture using local login approach. The local login approach has many problems, including

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

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

2.2.12 Desktop Login

We have not validated whether it is possible to implement TAS^3 architecture using desktop login approach. We recommend using one-time-passwords and the Authentication Service Protocol [?] to validate the authentication centrally using an IdP.

- Terminal servers: Mind-The-Box, Citrix, Windows TS, etc.

- Active Directory PDC

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

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

2.2.13 Fat Client Login

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

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

We recommend using one-time-passwords and the Authentication Service Protocol [**?**] to validate the authentication centrally using an IdP. One-time-passwords effectively solve the intermediary problem.

If Fat Client Login is a requirement, Liberty Advanced Client approach, see [**?**] and [**?**], SHOULD be used.

2.2.14 User Not Present or Batch Operations

TAS^3 specifies some approaches for doing this, see [**?**], mainly based on using advanced authorization to obtain discovery token without authenticating the User. Liberty Advanced Client approach, see [**?**] and [**?**], SHOULD be used.

2.3 Supported Identity Web Services Systems

The web services must satisfy some technical requirements

- Messages MUST be correlated, so each response is bound to request in an auditable way
  - Message ID correlation
  - Business Process Model and Instance IDs (or context or instance) to allow overarching correlation of several request-response pairs (e.g. to avoid actors who would have conflicts of interest overall that might not be identified when only working at level of individual request-response pairs)
    - PDP can receive this easy enough as an environment parameter and this is needed to support dynamic separation of duties
    - Gap: business process modelling does not express this?
    - Consider URL format hierarchical ID
    - Better typed, like LDAP DN format, or query string
- Requester and Responder MUST be identified (Req 10.4)
- Synchronous web service calls MUST be supported
- Asynchronous calls SHOULD be supported where needed. Business Process Engines will handle asynchrony.
- Subscribe - Notify mechanism SHOULD be supported where needed
  - subscription for events will be vital to pick up errors and notify of events like break the glass
  - subscribe and publish ws-eventing

  - Event bus as a subscribe and publish mechanism

- Maximum availability and use digital signature and encryption technologies, i.e. technical solutions to security and trust problems.

2.3.1 Framework

1. MUST support SOAP 1.2
2. MUST support XML-DSIG [?], a.k.a. RFC3275. In future we may introduce simpler schemes like Simple Web Token [?]. Using TLS connection stream as an audit trail element is impractical due to volume and inability of implementations to capture it. TLS stream as audit trail may also lead to inadvertent collateral disclosure.
3. MUST support Exclusive XML Canonicalization [?] for purposed of [?].
4. MAY support simple sign [?]. In future we will support Simple Web Token [?] which is very similar to simple sign.
5. MUST support XML-Enc [?] for protection of NameIDs and attributes, including bootstraps, as well as assertions, against an active intermediary. The common case in question is a SP that is about to make a web service call. To make such call, the SP must obtain from the discovery service a token that is passed to the web service provider. XML-Enc support allows the discovery service to pass in the encrypted token the pseudonym, and potentially some sensitive attributes, to the web service provider without the intermediary, SP in this case, being able to snoop on this confidential information. This case can not be solved using TLS alone as TLS is point-to-point and for this case TAS^3 architecture necessarily specifies an active intermediary.

2.3.2 Liberty ID-WSF Profile

1. MUST support ID-WSF 2.0 SOAP Binding [?] (this document is highly recommended reading).
2. MAY support ID-WSF 1.2
3. An implementation MUST support the following sec mechs, see [?]:

   - "urn:liberty:security:2005-02:TLS:Bearer"
   - "urn:liberty:security:2006-08:TLS:SAMLV2" (Holder-of-Key, HoK)

   A deployment MAY, as a configuration option, choose either.
4. MAY support following sec mechs for testing, but MUST NOT permit their use in production environments:

   - "urn:liberty:security:2005-02:null:Bearer"
   - "urn:liberty:security:2006-08:null:SAMLV2" (Holder-of-Key, HoK)
5. MAY support other TLS [?] based sec mechs, including ClientTLS.
6. MUST NOT permit non-TLS sec mechs in production environments
7. Implementations SHOULD be Liberty Alliance certified, see [?].
8. Implementations MUST support `<ProcessingContext>` "urn:liberty:sb:2003-08:ProcessingContext:Simulate" SOAP header and implement a "dry-run" feature using it. A deployment MAY, as a configuration option, enable this feature. Partially satisfies Reqs. *D1.2-12.13-Vfy* and *D1.2-12.16-OnlineTst*.

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

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

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

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

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

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

15. `<a:ReplyTo>` SOAP header MUST be included in all requests and MUST have value `http://www.w3.org/2005/03/a`

16. `<a:FaultTo>` SOAP header MUST NOT be supplied. All faults are sent to `<a:ReplyTo>` address, i.e. in the same HTTP request-response pair.

17. `<b:Sender>` SOAP header MUST be included in each web service message. [**?**] section 5.9, pp.21-22, is vague about when this is needed. To simplify matters we make it always mandatory.[1]

18. Request-Response message exchange patterm MUST be supported.

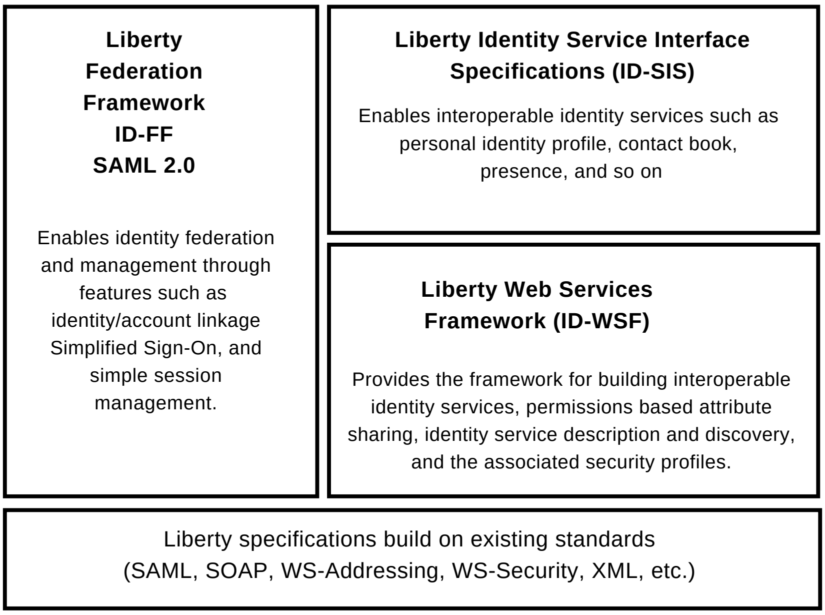

Liberty Federation Framework ID-FF SAML 2.0
Enables identity federation and management through features such as identity/account linkage Simplified Sign-On, and simple session management.
Liberty Identity Service Interface Specifications (ID-SIS)
Enables interoperable identity services such as personal identity profile, contact book, presence, and so on
Liberty Web Services Framework (ID-WSF)
Provides the framework for building interoperable identity services, permissions based attribute sharing, identity service description and discovery, and the associated security profiles.
Liberty specifications build on existing standards (SAML, SOAP, WS-Addressing, WS-Security, XML, etc.)

Figure 2.1: Liberty Alliance Architecture.

2.3.3 Bare WS-Security Header or Simplified ID-WSF

1. SHOULD NOT use, as many important security features such as message correlation, replay detection, and identification of endpoints are not supported by this mechanism.

2. Document resultant limitations if not implementing full ID-WSF.

[1] 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.

2.3.4 WS-Trust

- MAY support [?] in general, but MUST support if deploying the particular case of accessing external Credential Validation Service, per [?]

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

2.3.5 RESTful Approach

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

We have not validated whether it is possible to implement TAS^3 architecture using RESTful approach.

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

TAS^3 project plans to address RESTful binding in future work during 2010.

2.3.6 Message Bus Approach

We see deploying TAS^3 services on message bus architecture as feasible. This will be investigated in a future iteration of this deliverable.

2.4 Authorization Systems

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

Authorization systems are extensively covered in [?].

2.4.1 Authorization Queries

1. MUST support XACML 2.0 [?] request-response contexts for authorization queries
2. MAY support other versions of XACML
3. MAY support XACML policy language
4. MUST support XACML SAML Authorization Query extension [?] in order to allow policies to be dynamically passed to the PDP

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

2.4.2 Policy Languages

TAS^3 does not mandate any specific policy language. However, consider following possibilities:

1. PDP SHOULD support XACML 2.0 policy language [?]

2. PDP MAY support PERMIS 5.0 policy language
3. PDP MAY support P3P policy language
4. PDP MAY support PrimeLife privacy policies
5. PEP, PDP, and Obligations Service MAY support SOL1, see section 2.12, for obligations
6. CVS MAY support PERMIS Policy CVS Schema (cf. [?] Appendix 2)

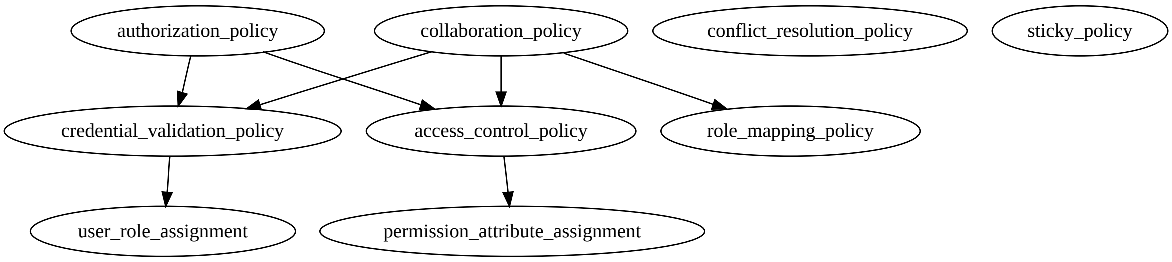


Figure 2.2: Hierarchy of policies

2.5 Trust and Security Vocabularies

Usage of ontologies in TAS^3 is thoroughly addressed in [?], which will map some of these vocabularies.

2.5.1 Levels of Authentication (LoA)

TAS^3 recommends the use of the NIST 4 levels of assurance as described in [?] and profiled in [?].

TAS^3 is working on determining whether and how to support LoA schemes of various European countries.

2.5.2 Vocabularies for Authorization

Some work has been done in RADIUS [?] and Diameter [?].

[?] is mainly about authentication, but authorization is also touched.

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

2.5.3 Vocabularies for Basic Attributes (PII)

Use of following vocabularies of PII is RECOMMENDED:

- LDAP inetOrgPerson [?]
- Liberty Personal Profile specification [?]
- X.500 standards, such as [?] and [?]. See also [?].

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

2.5.4 Discovery Vocabularies

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

Each Liberty service specifies its own Service Type value as well as a number discovery options. For example, see [?], [?], or [?].

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

2.5.5 Security and Trust Vocabularies

See [?] and [?] for a vocabulary of security mechanisms that MUST be used when applicable.

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

2.5.6 Audit Vocabularies

Audit events from RADIUS [?] and Diameter [?] are RECOMMENDED for use where applicable.

This section will be expanded in a future version of this document. As audit is active research topic, we benefit from the research during the TAS^3 project to specify this section in detail in the final version of thie document.

Specific Use of XDAS Fields

Specific Use of XDAS Event Numbers (really event codes)

2.6 Realization of the Discovery Function

- MUST support Liberty ID-WSF 2.0 Discovery Service specification [?]
- MAY support [?]
- MAY support UDDI, however this may require significant extensions to UDDI. Such extensions would need to be profiled.

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

2.7 Realization of the Credentials and Privacy Negotiator Function

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

2.7.1 Discovery in Credentials and Privacy Negotiation

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

2.7.2 Frontend Credentials and Privacy Negotiation

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

2.7.3 Components of Credentials and Privacy Negotiator

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).
2. SR passes the candidate list to the U-CPNA.
3. U-CPNA discovers the location of user’s attribute aggregator.
4. U-CPNA obtains a token with user’s pseudonym at the Attribute Aggregator.
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.

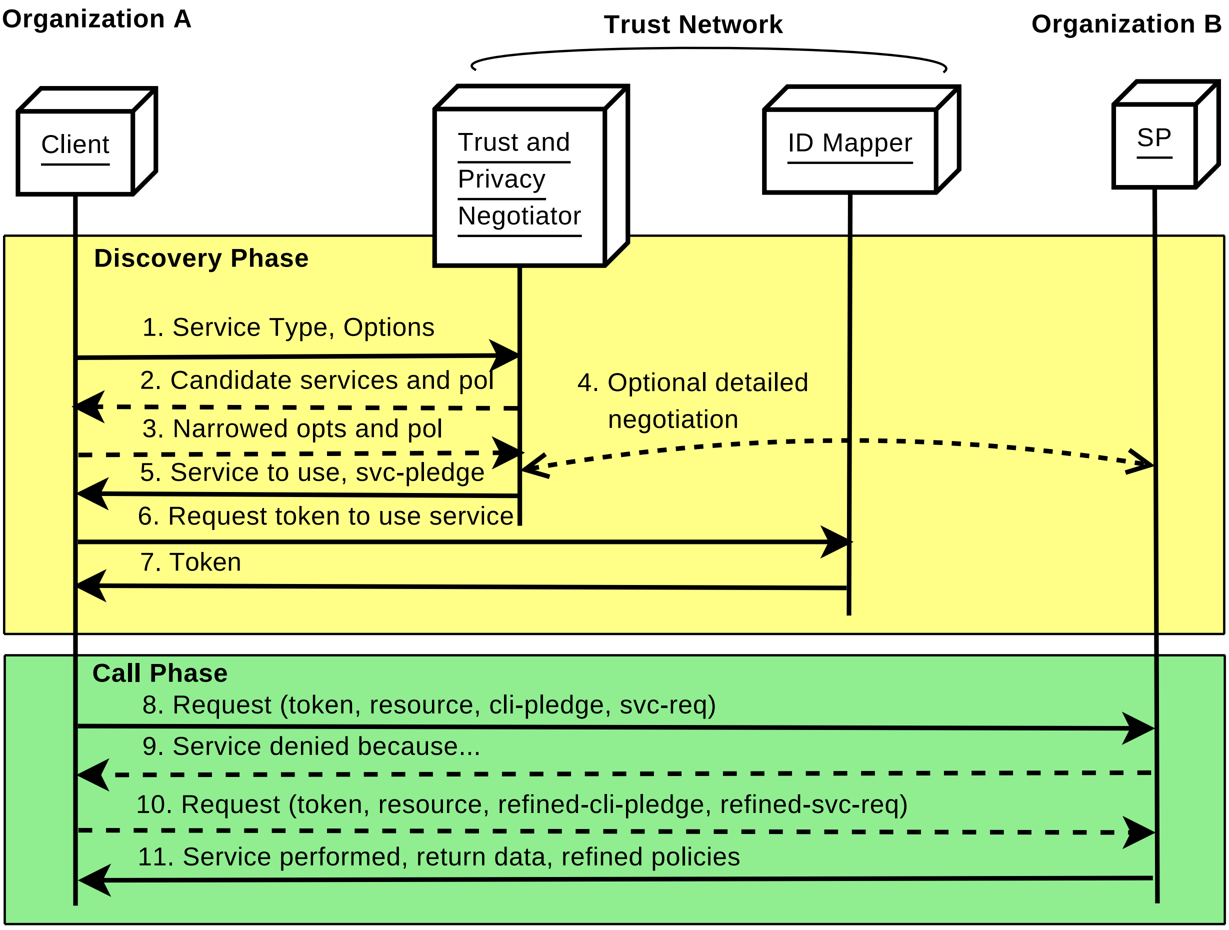

Organization A
Trust Network
Organization B
Client
Trust and Privacy Negotiator
ID Mapper
SP
Discovery Phase
1. Service Type, Options
2. Candidate services and pol
3. Narrowed opts and pol
4. Optional detailed negotiation
5. Service to use, svc-pledge
6. Request token to use service
7. Token
Call Phase
8. Request (token, resource, cli-pledge, svc-req)
9. Service denied because...
10. Request (token, resource, refined-cli-pledge, refined-svc-req)
11. Service performed, return data, refined policies

Figure 2.3: Credentials and Privacy Negotiation and Discovery steps

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

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

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

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

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

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

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

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

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

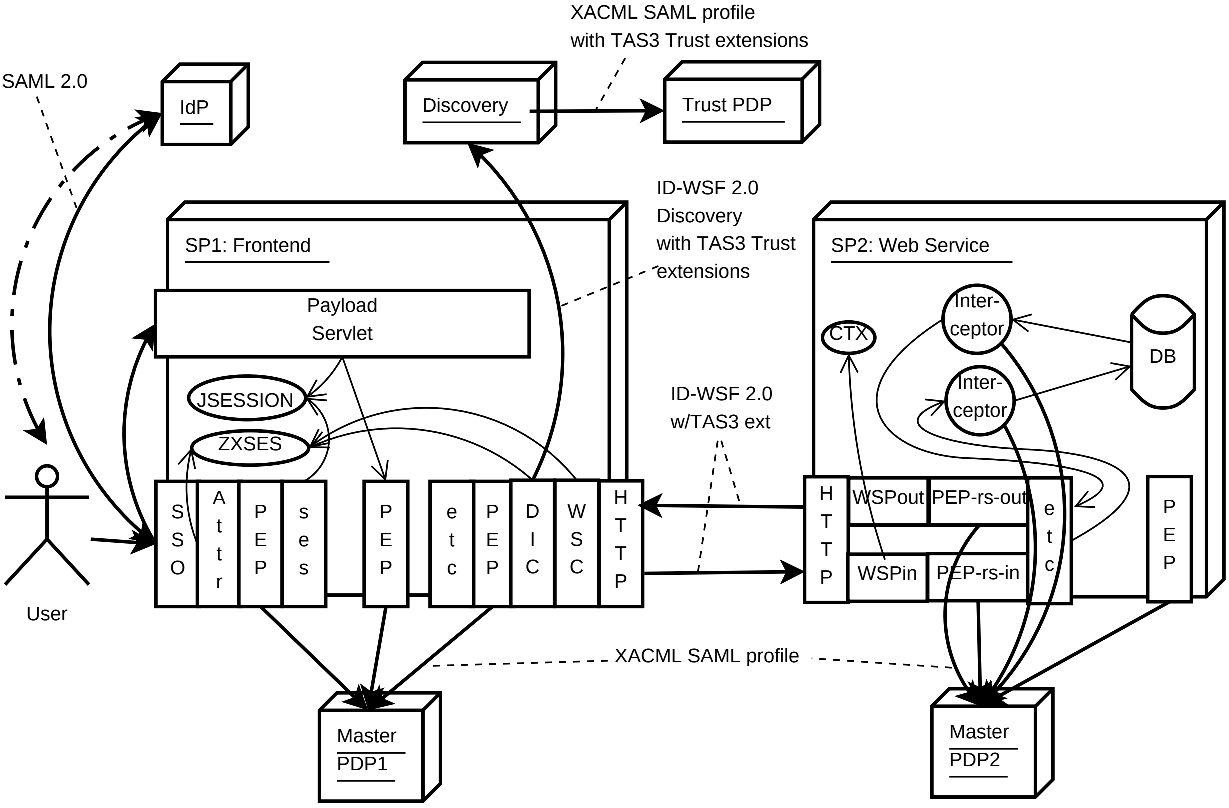


Figure 2.4: A deployment architecture for Credentials and Privacy Negotiation and Discovery

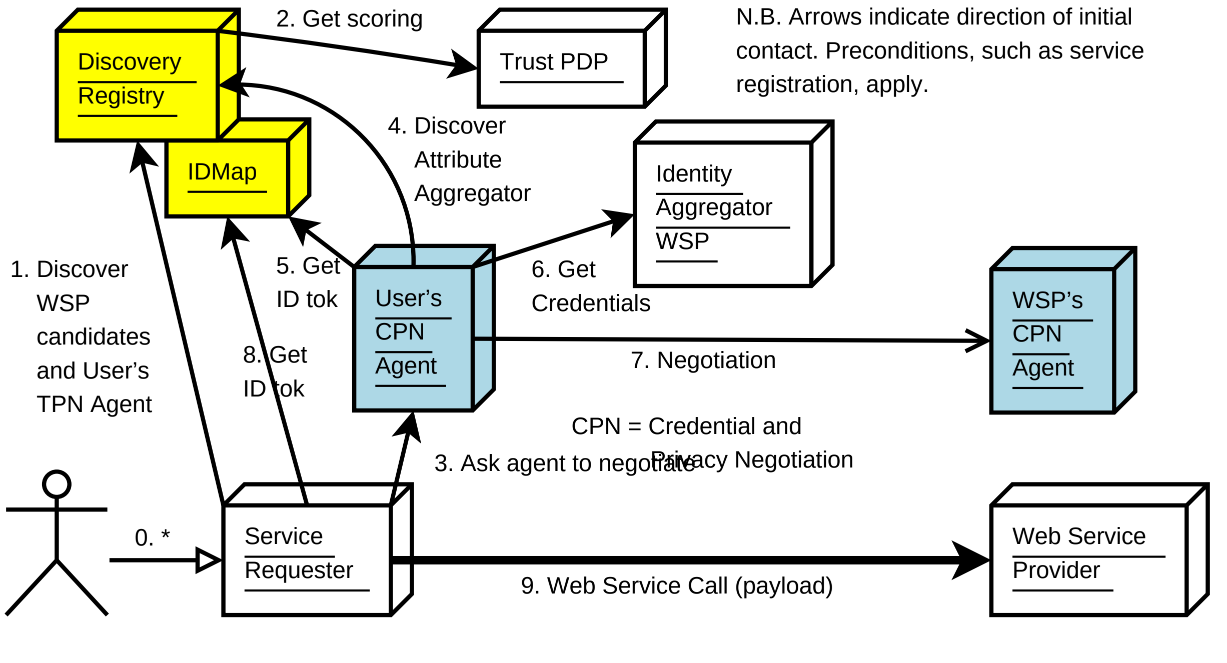


Figure 2.5: Credentials and Privacy Negotiation Components

531 **8** The discovery service returns to SR the EPR of the WSP. Finally the actual web service call is realized.

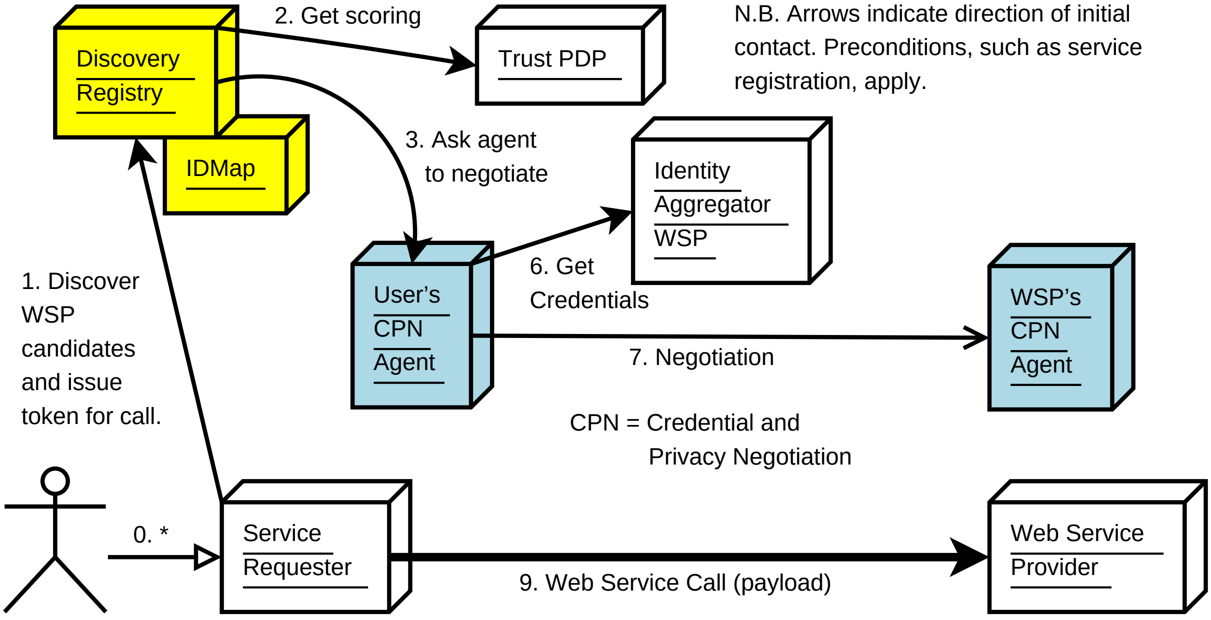

2. Get scoring
Discovery Registry
Trust PDP
N.B. Arrows indicate direction of initial contact. Preconditions, such as service registration, apply.
IDMap
3. Ask agent to negotiate
Identity Aggregator WSP
1. Discover WSP candidates and issue token for call.
User's CPN Agent
6. Get Credentials
WSP's CPN Agent
7. Negotiation
CPN = Credential and Privacy Negotiation
0. *
Service Requester
9. Web Service Call (payload)
Web Service Provider

Figure 2.6: Credentials and Privacy Negotiation optimized flow

2.7.4 Protocol between Service Requester and the Credentials and Privacy Negotiation Agent

Service Requester invokes the User's Credentials and Privacy Negotiation Agent as a regular web service. The body of the call needs to express the candidate (eventually candidate list to optimize better). Since discovery requests already express most of the interesting fields, we just wrap it in

```
<tas3cpn:CPNRequest>
  <di:RequestedService>
    <di:ServiceType>urn:x-foobar</di:ServiceType>
    <di:Framework version="2.0"></di:Framework>
  </di:RequestedService>
</tas3cpn:CPNRequest>
```

RequestedService identifies the NegotiationTarget, the resource, that the negotiation is about. Each interface can have its own way of identifying *resource(s)*. The NegotiationTarget includes specification of ServiceType as we assume that specification of the resource is interface specific.

Response will look like

```
  <tas3cpn:CPNResponse xmlns:tas3cpn="urn:tas3:cpn-agent">
    <lu:Status xmlns:lu="urn:liberty:util:2006-08" code="OK"></lu:Status>
    <tas3cpn:CPNRemoteReport>...</tas3cpn:CPNRemoteReport>
    <tas3cpn:CPNLocalReport>...</tas3cpn:CPNLocalReport>
    <tas3cpn:CPNChosenCredentialSet>...</tas3cpn:CPNChosenCredentialSet>
    <tas3cpn:CPNDisclosedCredentialSet>...</tas3cpn:CPNDisclosedCredentialSet>
  </tas3cpn:CPNResponse>
```

The <lu:Status> conveys whether negotiation was possible (e.g. whether aggregator could be contacted). OK value here does not indicate whether the actual negotiation process came to agreement. If

`<lu:Status>` is OK, the caller still needs to examine `<tas3cpn:CPNLocalReport>` (look for "no error" ***) to determine if agreement was reached.

The negotiation end point is obtained by using the ProviderID to lookup the service's extended SAML metadata and then extracting the end point from this metadata. (As a temporary testing kludge, negotiation end point can be constructed by taking the domain name of the ProviderID and using fixed "well known" port 9595. ***)

Complete CPN SOAP call looks like this:

```
  <e:Header>
    <a:MessageID e:actor="http://schemas.xmlsoap.org/soap/actor/next"
                 e:mustUnderstand="1"
                 wsu:Id="MID"
                 xmlns:a="http://www.w3.org/2005/08/addressing"
                 xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200\
401-wss-wssecurity-utility-1.0.xsd">urn:MNR-Cif7rlkmavkRm8cmPyRQh</a:Message\
ID>
    <a:To e:actor="http://schemas.xmlsoap.org/soap/actor/next"
          e:mustUnderstand="1"
          wsu:Id="TO"
          xmlns:a="http://www.w3.org/2005/08/addressing"
          xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss\
-wssecurity-utility-1.0.xsd">http://idp.tas3.pt:8081/zxididp?o=S</a:To>
    <b:Sender e:actor="http://schemas.xmlsoap.org/soap/actor/next"
              e:mustUnderstand="1"
              providerID="http://sp.tas3.pt:8080/zxidservlet/sso?o=B"
              wsu:Id="PRV"
              xmlns:b="urn:liberty:sb:2006-08"
              xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401\
-wss-wssecurity-utility-1.0.xsd" />
    <sbf:Framework e:actor="http://schemas.xmlsoap.org/soap/actor/next"
                   e:mustUnderstand="1"
                   version="2.0"
                   wsu:Id="FWK"
                   xmlns:sbf="urn:liberty:sb"
                   xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-2\
00401-wss-wssecurity-utility-1.0.xsd" />
    <wsse:Security e:actor="http://schemas.xmlsoap.org/soap/actor/next"
                   e:mustUnderstand="1"
                   wsu:Id="SEC"
                   xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-\
200401-wss-wssecurity-secext-1.0.xsd"
                   xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-2\
00401-wss-wssecurity-utility-1.0.xsd">
      <sa:EncryptedAssertion xmlns:sa="urn:oasis:names:tc:SAML:2.0:assertion\
">
        <xenc:EncryptedData Id="ED39"
                            Type="http://www.w3.org/2001/04/xmlenc#Element"
                            xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
          <ds:KeyInfo xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
            <ds:RetrievalMethod Type="http://www.w3.org/2001/04/xmlenc#Encry\
ptedKey"
                                URI="#EK39" />
          </ds:KeyInfo>
          <xenc:CipherData>
            <xenc:CipherValue>WlhfRE03eNxQ7rGH7
(snip)
4kdJYemjOMwd1zPVscag0NSwUABmeVusGJWh3yhiw+jLw==</xenc:CipherValue>
          </xenc:CipherData>
          <xenc:EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc\
#aes128-cbc" />
        </xenc:EncryptedData>
        <xenc:EncryptedKey Id="EK39"
                           xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
          <ds:KeyInfo xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
            <ds:X509Data>
              <ds:X509Certificate>MIICmDCCAgGgAwIBAgIEK3InjUPjt7
(snip)
FvsfT4RR6iA7KvTLs7yJRUDOmOpyAaSKy/5Mbd55fsatbYD5COIIlMN3IuU=
</ds:X509Certificate>
            </ds:X509Data>
          </ds:KeyInfo>
          <xenc:CipherData>
            <xenc:CipherValue>fAQTqq(snipdSI=</xenc:CipherValue>
          </xenc:CipherData>
          <xenc:EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc\
#rsa-1_5" />
          <xenc:ReferenceList>
            <xenc:DataReference URI="#ED39" />
          </xenc:ReferenceList>
        </xenc:EncryptedKey>
      </sa:EncryptedAssertion>
      <wsse:SecurityTokenReference>
        <wsse:KeyIdentifier ValueType="http://docs.oasis-open.org/wss/oasis-\
wss-saml-token-profile-1.1#SAMLID" />
      </wsse:SecurityTokenReference>
      <wsu:Timestamp wsu:Id="TS">
        <wsu:Created>2009-12-19T11:33:57Z</wsu:Created>
      </wsu:Timestamp>
    </wsse:Security>
  </e:Header>
  <e:Body>
    <tas3cpn:CPNRequest xmlns:tas3cpn="urn:tas3:cpn-agent">
      <di:RequestedService xmlns:di="urn:liberty:disco:2006-08">
        <di:Framework version="2.0" />
        <di:ServiceType>urn:x-foobar</di:ServiceType>
        <di:ProviderID>http://wsp.tas3.pt:8080/wsp?o=B</di:ProviderID>
        <di:Action>urn:x-foobar:Create</di:Action>
      </di:RequestedService>
    </tas3cpn:CPNRequest>
  </e:Body>
</e:Envelope>
```

You can easily generate a test request with following shell script:

```
zxcall -a https://idp.tas3.eu/zxididp?o=B bh:betty -t urn:tas3:cpn-agent <<X\
ML
<tas3cpn:CPNRequest xmlns:tas3cpn="urn:tas3:cpn-agent">
```

```
  <di:RequestedService xmlns:di="urn:liberty:disco:2006-08">
    <di:ServiceType>urn:x-foobar</di:ServiceType>
    <di:ProviderID>http://wsp.tas3.pt:8080/wsp?o=B</di:ProviderID>
    <di:Framework version="2.0" />
    <di:Action>urn:x-foobar:Create</di:Action>
  </di:RequestedService>
</tas3cpn:CPNRequest>
```

XML

2.7.5 Protocol between Credentials and Privacy Negotiation Agent and Attribute Aggregator

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

2.7.6 Protocol between Credentials and Privacy Negotiation Agent and Service

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

i. the one used by TrustBuilder 2 [?]

ii. one based on the Web Service Profile of XACML [?] as enhanced by [?]

iii. one based on an enhanced Liberty Discovery Service [?]

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

2.8 Using Trust Scoring in Discovery

When making discovery call, the minimum acceptable trust level SHOULD be conveyed as discovery option. The discovery service will then filter the candidates by calling Trust PDP and looking at the Permit / Deny response.

2.8.1 Specifying Trust Inputs

See D5.4 section 3.2 "Installation and Configuration Instruction" for full description of Trust Inputs and in particular specifying policies that capture trust inputs.

The trust inputs are specified as discovery options, e.g.

```
urn:tas3:trust:input:ctl1:policyid=ABC
urn:tas3:trust:input:ctl1:ranking=avgfeedback
urn:tas3:trust:input:ctl1:ranking=oct
```

where "ctl1" identifies the input as conformant to Combined Trust Language version 1 and "policyid=ABC", "ranking=oct", etc., are the trust language specific parameters.

The Discovery service will pass the discovery options to the Trust PDP as XACML environment attributes as follows:

```
  <xasp:XACMLAuthzDecisionQuery ID="RmQtc_SvgPVYANCPrELYfjl59"
                                IssueInstant="2009-12-19T11:33:54Z"
                                Version="2.0"
```

```
                                    xmlns:xasp="urn:oasis:xacml:2.0:saml:protocol:schema:os">
      <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">...</>
      <sa:Issuer xmlns:sa="urn:oasis:names:tc:SAML:2.0:assertion">http://sp.tas3.pt:8080/zxidse
      <xac:Request xmlns:xac="urn:oasis:names:tc:xacml:2.0:context:schema:os">
        <xac:Action>
          <xac:Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
                         DataType="http://www.w3.org/2001/XMLSchema#string">
            <xac:AttributeValue>Show</xac:AttributeValue>
          </xac:Attribute>
        </xac:Action>
        <xac:Environment>
          <xac:Attribute AttributeId="urn:tas3:trust:input:ctl1:policyid"
                         DataType="http://www.w3.org/2001/XMLSchema#string">
            <xac:AttributeValue>ABC</xac:AttributeValue>
          </xac:Attribute>
          <xac:Attribute AttributeId="urn:tas3:trust:input:ctl1:ranking"
                         DataType="http://www.w3.org/2001/XMLSchema#string">
            <xac:AttributeValue>avgfeedback</xac:AttributeValue>
          </xac:Attribute>
          <xac:Attribute AttributeId="urn:tas3:trust:input:ctl1:ranking"
                         DataType="http://www.w3.org/2001/XMLSchema#string">
            <xac:AttributeValue>oct</xac:AttributeValue>
          </xac:Attribute>
        </xac:Environment>
        <xac:Resource>...</xac:Resource>
        <xac:Subject>...</xac:Subject>
      </xac:Request>
    </xasp:XACMLAuthzDecisionQuery>
```

Please note that the `policyid` refers to a policy that has been precreated at the Trust PDP and that expresses minimum values for the various trust parameters.

In terms of API the values would be passed as follows (line has been wrapped before ampersands for readability):

```
  epr = tas3_get_epr(cf, ses, "urn:service:type", null,

"urn:tas3:trust:ctl1:input:policyid=ABC

         &urn:tas3:trust:ctl1:input:ranking=avgfeedback
         &urn:tas3:trust:ctl1:input:ranking=oct",

        "Show", 1);
```

Calling *tas3_get_epr()* allows user interface with trust scorings to be presented. If this is not of interest, the discovery options can be given directly to *tas3_call()* function:

```
  ret = tas3_call(cf, ses, "urn:service:type", null,

"urn:tas3:trust:ctl1:input:policyid=ABC

         &urn:tas3:trust:ctl1:input:ranking=avgfeedback
         &urn:tas3:trust:ctl1:input:ranking=oct",

        null, "<Request/>");
```

A way to test Trust negotiation from command line is

```
 ./zxcall -d -a https://idp.tas3.eu/zxididp?o=B bh:betty -t urn:tas3:karlsruhe:test:service-di
```

2.8.2 Returning Trust Scores

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

Example

```
<StatusCode Value="urn:oasis:names:tc:xacml:1.0:status:ok">
  <StatusCode Value="urn:tas3:trust:ctl1:ranking:avgfeedback=0.960922">
    <StatusCode Value="urn:tas3:trust:ctl1:ranking:oct=0.711221"/>
  </StatusCode>
</StatusCode>
```

The status codes are extracted by the Discovery Service and packaged as additional EPR metadata when returned to the caller:

```
<a:EndpointReference
    xmlns:a="http://www.w3.org/2005/08/addressing"
    xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0
    notOnOrAfter="2037-01-05T23:03:59.001Z"
    wsu:Id="EPRID921FPo3ZNEt_3rHtJFoU">
  <a:Address>
    http://141.26.143.22:8080/matching-simple/services/SimpleResource
  </a:Address>
  <a:Metadata>
    <sbf:Framework
        xmlns:sbf="urn:liberty:sb"
        version="2.0"/>
    <di:Abstract xmlns:di="urn:liberty:disco:2006-08">Test</>
    <di:ProviderID xmlns:di="urn:liberty:disco:2006-08">
      http://141.26.143.22:8080/wspdemosp3.xml
    </di:ProviderID>
    <di:ServiceType xmlns:di="urn:liberty:disco:2006-08">urn:tas3:matchingservice</>
    <tas3:Trust vers="ctl1">
      <tas3:TrustRanking metric="avgfeedback" val="0.960922"/>
      <tas3:TrustRanking metric="oct" val="0.711221"/>
    </tas3:Trust>
  </a:Metadata>
</a:EndpointReference>
```

2.9 Realization of the Audit and Dashboard Function

2.9.1 Audit Event Bus

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

Tentative protocol choice (in order of preference):

1. AMQP [**?**]
2. Liberty Accounting Service [**?**] with subscriptions and notifications [**?**] and [**?**].
3. Diameter [**?**]
4. RADIUS [**?**]
5. Apache Muse

Whichever transport is chosen, the actual audit records are packaged as OpenXDAS messages (see: `openxdas.sourceforge.net`).

2.9.2 Audit Event Ontology

- Enumeration of mandatory edit events according to some standard
    - RADIUS and Diameter communities have defined at least some messages
- ZXID logging documentation [**?**] provides an idea, at least applicable to SSO

2.9.3 Dashboard Function

- Dashboard should also realize the "PII Consent Service" or "Privacy Manager" at large.
- SHOULD support Liberty Interaction service [**?**]

2.9.4 User Interaction

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

a. Liberty Interaction service [**?**] where a web services call is made to the interaction service. This service is often colocated with the Dashboard.

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

Special attribute for interaction iFrame URL.

2.9.5 TAS^3 User Interaction Widget

TAS^3 Widget is a special user interaction device inserted into SP web sites (e.g. by means of iFrame), but pulled from the Dashboard.

The widget will refresh itself periodically from the dashboard and if necessary solicit interaction from the user. In many ways it is similar to web based instant messaging client.

If a WSP wants to interact with a user, it discovers the location of user's interaction service. Typically this will point to the dashboard. When interaction request is sent, the dashboard queues it to be delivered on the next refresh of the widget. When user replies, the interaction service call is completed and reslt returned to the WSP.

The URL for loading the widget to the SP user interface is determined either by an attribute passed on SSO or by discovering a special Widget resource.

The SSO attribute is named

```
urn:tas3:uiwidget:epr
```

The service type for discovery is called

```
urn:tas3:uiwidget
```

Other actions that can be integrated into the widget (so that the web page does not have to implement them separately):

- Single logout

2.10 Realization of Delegation Function

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

2.11 Attribute Authorities

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

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

There are several ways of implementing a data web service. [?] specifies AttributeQuery protocol, but does not adequately specify the transport binding and peer authentication. TAS^3 attribute authority SHOULD support [?] AttributeQuery protocol using TAS^3 SOAP binding, see section 2.3.2.

Other data web services, such as ID-DAP [?] over TAS^3 SOAP binding, MAY be supported. A deployment may also make local or proprietary arrangements for accessing a non TAS^3 attribute authority, e.g. using LDAP [?] or WebDAV with file containing attribute certificate or SAML attribute assertion.

2.12 TAS^3 Simple Obligations Language (SOL)

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

In general the obligations and sticky policies can be expressed in any convenient language. Unfortunately no standard language has emerged in the industry for this type of application despite many being proposed. TAS^3 is committed to supporting multiple such languages, but for purposes of pilots and other simple applications we define "TAS^3 Simple Obligations Language n^{o}1" (SOL1) with potential future versions to follow.

SOL obligations MAY be used in XACML obligations as described in [?]. In particular, D7.1 Appendix A1.2 provides an example. In short, they MUST appear in an `Obligation/AttributeAssignment` element. When passed in `<b:UsageDirective>`, `<xa:Obligation>` element MUST be used as a wrapper. Use of `<xa:Obligation>` element as a wrapper in other XML contexts is RECOMMENDED.

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

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

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

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

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

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

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

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

Example

```
<b:UsageDirective id="USE">
  <xa:Obligation ObligationId="urn:tas3:sol1" FulfillOn="Permit">
    <xa:AttributeAssignment
        AttributeId="urn:tas3:sol1:pledge"
        DataType="http://www.w3.org/2001/XMLSchema#string">
          urn:tas3:sol:vers=1
          urn:tas3:sol1:delon=1255555377
          urn:tas3:sol1:use=urn:tas3:sol1:use:forpurpose
          urn:tas3:sol1:share=urn:tas3:sol1:share:group
          urn:tas3:sol1:repouse=urn:tas3:sol1:repouse:oper
    </xa:AttributeAssignment>
  </xa:Obligation>
</b:UsageDirective>
```

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

2.12.1 SOL1 Query String Attributes

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

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

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

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

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

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

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

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

`urn:tas3:sol1:use:forpurpose` (3) Information will be used only for the purpose it was collected, in abstract. This usage is discouraged. Instead the specific purpose should be specified using format

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

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

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

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

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

`urn:tas3:sol1:use:appident` (7) Information can be used by the application towards other purposes (user may be identified)

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

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

`urn:tas3:sol1:use:mktanon` (10) Information can be used by the organization for marketing purposes as long as the user is not explicitly identified

`urn:tas3:sol1:use:mktident` (11) Information can be used by the organization for marketing purposes (user may be identified)

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

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

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

`urn:tas3:sol1:use:grpmktident` (15) Information can be used within the business group for marketing purposes (user may be identified)

`urn:tas3:sol1:use:shareanon` (16) Information can be shared with anyone for other nonmarketing purposes as long as the user is not explicitly identified

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

`urn:tas3:sol1:use:sharemktanon` (18) Information can be shared with anyone for marketing purposes as long as user is not explicitly identified

`urn:tas3:sol1:use:sharemktident` (19) Information can be shared with anyone for marketing purposes (user may be identified)

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

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

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

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

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

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

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

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

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

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

- **`urn:tas3:sol1:repouse:never`** No need to report use (seldom appears)
- **`urn:tas3:sol1:repouse:all`** Report any and all use
- **`urn:tas3:sol1:repouse:oper`** Report operational use, but not statistical or administrative use.
- **`urn:tas3:sol1:repouse:stat:immed`** Report use in near real time. for day need to be reported, if there was any use.
- **`urn:tas3:sol1:repouse:stat:daily`** No need to report individual use, but summary statistics for day need to be reported, if there was any use.
- **`urn:tas3:sol1:repouse:stat:weekly`** No need to report individual use, but summary statistics for week need to be reported, if there was any use.
- **`urn:tas3:sol1:repouse:stat:monthly`** No need to report individual use, but summary statistics for month need to be reported, if there was any use.
- **`urn:tas3:sol1:repouse:stat:quarterly`** No need to report individual use, but summary statistics for quarter (last 3 months) need to be reported, if there was any use.
- **`urn:tas3:sol1:repouse:stat:semestral`** No need to report individual use, but summary statistics for semester (last 6 months) need to be reported, if there was any use.
- **`urn:tas3:sol1:repouse:stat:yearly`** No need to report individual use, but summary statistics for year need to be reported, if there was any use.

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

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

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

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

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

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

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

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

`urn:tas3:sol1:contract` Contract identifier.

`urn:tas3:sol1:contract-sub` Subcontract or amendment identifier

`urn:tas3:sol1:contract-part` Part, exhibit, annex, or clause identifier.

2.12.2 Matching Pledges to Sticky Policies and Obligations

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

Example

Consider the following request

```
<e:Envelope>
  <e:Header>
    <!-- WS-Addressing headers and wsse:Security with DSIG not shown -->
    <b:UsageDirective id="USE">
      <xa:Obligation ObligationId="urn:tas3:sol1" FulfillOn="Permit">
        <xa:AttributeAssignment
            AttributeId="urn:tas3:sol1:pledge"
            DataType="http://www.w3.org/2001/XMLSchema#string">
          urn:tas3:sol:vers=1
          urn:tas3:sol1:delon=1255555377
          urn:tas3:sol1:use=urn:tas3:sol1:use:purpose
          urn:tas3:sol1:share=urn:tas3:sol1:share:group
          urn:tas3:sol1:repouse=urn:tas3:sol1:repouse:oper
       </>
      </>
    </>
  </>
  <e:Body id="BDY">
    <idhrxml:Query>...</></></>
```

Now, backend returns the following data

```
<dataItem id="1">
  <tas3sol:Obligations xmlns:tas3sol="http://tas3.eu/tas3sol/200911/">
    urn:tas3:sol:vers=1
    urn:tas3:sol:delon=1255555378
```

```
      urn:tas3:sol1:use=urn:tas3:sol1:use:transaction
    </>
    <data>value</>
  </>

  <dataItem id="2">
    <tas3sol:Obligations xmlns:tas3sol="http://tas3.eu/tas3sol/200911/">
      urn:tas3:sol:vers=1
      urn:tas3:sol:delon=1255555376
      urn:tas3:sol1:use=urn:tas3:sol1:use:purpose
      urn:tas3:sol1:repouse=urn:tas3:sol1:repouse:all
    </>
    <data>value</>
  </>

  <dataItem id="3">
    <tas3sol:Obligations xmlns:tas3sol="http://tas3.eu/tas3sol/200911/">
      urn:tas3:sol:vers=1
      urn:tas3:sol:delon=1255555378
      urn:tas3:sol1:use=urn:tas3:sol1:use:purpose
      urn:tas3:sol1:repouse=urn:tas3:sol1:repouse:oper,repouse=urn:tas3:sol1:repouse:stat:weekl
    </>
    <data>value</>
  </>
```

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

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

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

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

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

2.12.3 Passing Simple Obligations Dictionaries Around

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

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

Example

```
<e:Envelope>
  <e:Header>
    <!-- WS-Addressing headers and wsse:Security with DSIG not shown -->
    <b:UsageDirective id="USE">
      <xa:Obligation ObligationId="urn:tas3:sol1" FulfillOn="Permit">
        <xa:AttributeAssignment
            AttributeId="urn:tas3:sol1:pledge"
            DataType="http://www.w3.org/2001/XMLSchema#string">
          urn:tas3:sol:vers=1
          urn:tas3:sol1:delon=1255555377
          urn:tas3:sol1:use=urn:tas3:sol1:use:purpose
          urn:tas3:sol1:share=urn:tas3:sol1:share:group
          urn:tas3:sol1:repouse=urn:tas3:sol1:repouse:oper
        </>
      </>
      <tas3sol:Dict xmlns:tas3sol="http://tas3.eu/tas3sol/200911/">
        Entities:
          Data Subject (Agent the Data describes)
          Data Processor (Agent that processes the Data)
          Data (Information which is a resource under protection)
          Organisation (a Data Processor)
          Marketing (an Action)
          Process (an Action of manipulating Data)

        Relations:
          Identify
          Retain

        Property
          May (property of an action)
          Must (property of an action)

        urn:tas3:sol1:use:mktident is an enumerator of urn:tas3:sol1:use

        urn:tas3:sol1:use:mktident means
          Organization (who) - Process (action) - Data (what) - Marketing (why)
          Organization (who) - Identify (action) - Data Subject (What)
      </>
    </>
  </>
  <e:Body id="BDY">
    <idhrxml:Query>...</></></>
```

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

2.13 Realization of Sticky Policies

As discussed in [**?**] section 4.1 "Protocol Support for Conveyance of Sticky Policies", Encapsulating Security Layer (ESL) is one approach for implementing sticky policies. While total encapsulation is possible, for already established applications protocols something lighter weight is desired. Most properties of ESL can also be implemented by a special SOAP header that references all the elements that would have been contained or referenced by the ESL approach. The subtle, but salient, diffenrence 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 [**?**].

Example

```
<e:Envelope>
  <e:Header>
    <wsse:Security>...(signature here to bind ESLPolicies and Body)...</>
    <tas3:ESLPolicies mustUnderstand="1">
      <tas3:ESLApply>
        <tas3:ESLRef ref="#data1"/>
        <tas3:ESLRef xpath="container/subcontainer"/>
        <xa:Obligation ObligationId="urn:tas3:sol1">
          <xa:AttributeAssignment
              AttributeId="urn:tas3:sol1:require"
              DataType="http://www.w3.org/2001/XMLSchema#string">
            urn:tas3:sol:vers=1
            urn:tas3:sol1:delon=1255555377
          </xa:AttributeAssignment>
        </xa:Obligation>
      </tas3:ESLApply>
      <tas3:ESLApply>
        <tas3:ESLRef ref="#data2"/>
        <xa:Obligation ObligationId="urn:tas3:sol1">
          <xa:AttributeAssignment
              AttributeId="urn:tas3:sol1:require"
              DataType="http://www.w3.org/2001/XMLSchema#string">
            urn:tas3:sol:vers=1
            urn:tas3:sol1:delon=1255566666
          </xa:AttributeAssignment>
        </xa:Obligation>
      </tas3:ESLApply>
    </tas3:ESLPolicies>
  </e:Header>
  <e:Body>
    <data id="data1" value="foo">
    <data id="data2" value="bar">
    <container>
       <subcontainer value="goo"/>
    </container>
  </e:Body>
</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.

2.14 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.

```
<e:Envelope>
  <e:Header>
    <wsse:Security>...</>
    <tas3:Credentials xmlns:tas3="http://tas3.eu/tas3/200911/">
      ... reuse XACML or SAML attribute schema
    </tas3:Credentials>
  </e:Header>
  <e:Body>...</>
</e:Envelope>
```

2.15 Uniform Application Status and Error Reporting

Traditionally Web Service application protocols have defined their own error and status reporting mechanisms. TAS^3 standardizes the status reporting by adding a standardized SOAP header that the application SHOULD insert if it wishes to enable some automatic TAS^3 processing. This is especially important for automation of Online Compliance Testing.

Some ways the errors can be reported

1. Network or socket layer, e.g. drop the connection in case of a security violation. This is very extreme response and SHOULD NOT be used normally, unless there is a genuine threat, such as suspected Denial-of-Service (DoS) attack.

2. HTTP layer error codes. In normal operation, 200 should be used. In particular 4xx and 5xx codes SHOULD NOT be used to indicate authorization errors deep in the application or application errors. The HTTP error codes SHOULD generally be used for errors that are detected at web server level.

3. Application platform errors, such as stack backtraces, SHOULD NOT happen. All errors SHOULD be trapped and appropriately reported by the application. Despite this rule, the reality of application development means that stack traces will be output by buggy or immature software.

4. SOAP faults. Generally SOAP faults should only be used to indicate SOAP transport level errors, as defined by SOAP and ID-WSF specifications.

   The API, such as *tas3_get_fault()*, for creating and inspecting TAS^3 related SOAP faults is described in section 3.1.13 "SOAP Fault and Status Generation and Inspection".

5. ID-WSF special headers. Some ID-WSF level errors cause an ID-WSF specific SOAP headers to be emitted in the response.

6. TAS^3 error header SHOULD be used to report all TAS^3 and application level errors.

7. Application level error mechanisms MAY be used to report application level errors. It is RECOMMENDED that the application level protocols be designed to use the TAS^3 error headers or at least the Liberty Utility schema dedined `<Status>` element [**?**].

2.15.1 TAS^3 Status Header

The TAS^3 Status Header is based on the `<Status>` element defined in Liberty Utility Schema, see [**?**].

```
<e:Envelope>
  <e:Header>
    <tas3:Status
```

```
         xmlns:tas3="http://tas3.eu/tas3/200911/"
         ctlpt="urn:tas3:ctlpt:app"
         code="OK"/>
     </e:Header>
     <e:Body>...</>
   </e:Envelope>
```

The API, such as *tas3_get_tas3_status()* for creating and inspecting TAS^3 Status Header is described in section 3.1.13 "SOAP Fault and Status Generation and Inspection".

2.15.2 TAS^3 Status Codes

The `code` XML attribute may contain any of the ID-WSF defined status codes, see [**?**] Table 2 on pp.12-13, including the special value "OK" to indicate success. It may also contain any application specific status indications, provided that they are qualified to their own namespace using URN or URL constructs. Finally it may contain any of the following TAS^3 defined status codes:

`urn:tas3:status:deny` Operation denied by authorization layer

`urn:tas3:status:notapplicable` Operation not applicable from authorization perspective

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

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

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

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

2.15.3 TAS^3 Control and Reporting Points

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

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

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

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

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

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

2.16 Registration of Business Process Models

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

3 The Official TAS^3 API (normative, but non-exclusive)

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

The TAS^3 API described herein is meant to have multiple implementations. Each implementation provides

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

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

The particular API in use is chosen by the programmer by including the appropriate header file or interface description. The particular API implementation in use is chosen by the system administrator or the programmer by linking against a particular library providing the TAS^3 binary interface, or by dynamically loading a module implementing the said binary interface. This leaves great implementation flexibility while accurately describing the TAS^3 interface and implementation at source code (API) and binary (ABI) level.

3.1 Language Independent Description of the API

Since all language specific bindings, by-and-large, share the same semantics, the functions and methods are first described generically, using pseudocode if needed. Each language binding takes the same parameters and behaves in the way that API would naturally work, *mutantis mudandis*, for that language.[1]

The five essential APIs are

tas3_sso() SSO (with optional application independent authorization)

tas3_az() Application Dependent Authorization

tas3_call() Web Services Client: call a web service and validate response

tas3_wsp_validate() Validate that web service request can be processed

tas3_wsp_decorate() Create a web service response

3.1.1 Single Sign On (SSO) Alternatives

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

[1] 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.

Some alternatives for supporting SSO:

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

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

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

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

There are many options. Most of these have sensible default values or can be specified in a configuration file. The first parameter either is a configuration object, or a configuration string that modifies or adds to the 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 *exit(2)*, 0=return "n", even if auto CGI
2	0x02	TAS3_AUTO_REDIR	Automatic. handle redirects, assume CGI (calls *exit(2)*)
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	0xfff	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

Example Usage

```
res = tas3_sso(conf, request['QUERY_STRING'], 0x1800);
switch (substr(res, 0, 1)) {
case 'L': header(res); return 0; # Redirect
case 'n': return 0;              # already handled
case 'b': return my_send_metadata();
case 'e': return my_render_idp_selection_screen();
case 'd': return my_start_session_and_render_protected_content();
default:  error_log("Unknown tas3_sso() res(%s)", res); return 0;
}
```

Return values

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

L Redirection request (L as in Location header). The full contents of the res is the redirection request, ready to be printed to stdout of a CGI. If you want to handle the redirection some other way, you can parse the string to extract the URL and do your thing. This res is only returned if you did not set TAS3_AUTO_REDIR.

Example:

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

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

Example:

```
CONTENT-TYPE: text/html

<title>Login page</title>
...
```

Example (metadata):

```
CONTENT-TYPE: text/xml

<m:EntityDescriptor>
...
```

Less than ("<") Content without headers. This could be HTML content for login page or metadata XML. To know which (and set content type correctly), you would have to parse the content. This res format is only applicable if you did not specify TAS3_AUTO_CTYPE (but did specify TAS3_AUTO_CONTENT).

n Do nothing. The operation was somehow handled internally but the *exit(2)* was not called (e.g. TAS3_AUTO_SOAP was NOT specified). The application should NOT attempt generating any output.

b Indication that the application should send SP metadata to the client. This res is only returned if you did not set TAS3_AUTO_META.

c Indication that the application should send SP CARML declaration to the client. This res is only returned if you did not set TAS3_AUTO_META.

e Indication that the application should display the IdP selection page. This res is only returned if you did not set TAS3_AUTO_CONTENT.

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

```
dn: idpnid=Pa45XAs2332SDS2asFs,affid=https://idp.demo.com/idp.xml
objectclass: zxidsession
affid: https://idp.demo.com/idp.xml
idpnid: Pa45XAs2332SDS2asFs
authnctxlevel: password
sesid: S12aF3Xi4A
cn: Joe Doe
```

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

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

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

3.1.3 Authorization: decision = *tas3_az(conf, qs, ses)*

Implicit application independent authorization steps are performed in *tas3_sso()* SSO, *tas3_call()* Service Requester, *tas3_wsp_validate()*, and *tas3_wsp_decorate()* APIs. To activate them, you need to supply appropriate configuration options. Specifics of this configuration are implementation dependent.

The *tas3_az()* function is the main work horse for requesting authorization decisions from the PDPs. It allows programmer to make Application Dependent authorization calls, supplying some or all of the attributes needed in a XACML request. *tas3_az()* can also use attributes from the session, if configured. Specifics of this configuration are implementation dependent.

conf the configuration string or object

qs if supplied, any CGI variables are imported to session environment as attributes according to configuration. Format is CGI Query String.

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

return 0 if deny (for any reason, e.g. indeterminate), or string representation of `<xac:Response>` element if permit

Example Pseudocode

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

3.1.4 Authorization base: decision = *tas3_az_base(conf, qs, ses)*

This is similar to *tas3_az()* with the difference that the `<xac:Response>` element is returned even in the deny and indeterminate cases (null is still returned if there was an error). Effectively this *base* form does not make judgement about whether `<xac:Response>` means permit, deny, or something else.

You should use this function if the Deny message contains interesting obligations (normally it does not).

3.1.5 Web Service Call: ret_soap = *tas3_call(cf, ses, svctype, url, di_opt, az_cred, req_soap)*

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

Similarly on return, *tas3_call()* will check all TAS^3 relevant SOAP headers and signatures, but will still return the entire SOAP envelope as a string so that the application layer can, if it wants, look at the headers.

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

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

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

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

After executing the SOAP call and verifying any returned TAS^3 relevant headers and signatures, *tas3_call()* may contact a PDP to authorize receiving data, and to pass on any obligations that were received. This corresponds to application independent *Requester In PEP* and is configurable: you can disable it if you prefer to make explicit application dependent call to *tas3_az()*. The contents of the XACML request are determined based on the response, session, `az_cred` parameter, which is shared for both Responder Out and Responder In PDP calls, and configuration, which is implementation dependent.

cf Configuration object, see *tas3_new_conf_to_cf()*

ses Session object, used to locate EPRs, see *tas3_new_ses()*

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

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

di_opt (Optional) Additional discovery options for selecting the service, query string format

az_cred (Optional) Additional authorization credentials or attributes, query string format. These credentials will be populated to the session's attribute pool in addition to the ones obtained from SSO and other sources. Then a PDP is called to get an authorization decision (as well as obligations we pledge to support). This implements generalized (application independent) Requester Out and Requester In PEPs. To implement application dependent PEP features you should call *tas3_az()* directly.

req_soap string used as SOAP body or as SOAP envelope template.

return SOAP envelope as a string.

Example

```
env = tas3_callf(cf, ses, "urn:hrxml:idhrxml", 0,0,0,
        "<idhrxml:Modify>"
          "<idhrxml:ModifyItem>"
            "<idhrxml:Select>%s</idhrxml:Select>"
            "<idhrxml:NewData>%s</idhrxml:NewData>"
          "</idhrxml:ModifyItem>"
        "</idhrxml:Modify>", cgi.select, cgi.data);
if (env) {
  xml = xml_parse(env);
  if (xml->Status->code == "OK") {
    INFO("Data is " + xml->Data);
  } else {
    ERR("Web service error " + xml->Status->code);
  }
} else {
  ERR("HTTP failure");
}
```

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

3.1.6 Requester out: req_decor_soap = *tas3_wsc_prepare_call(cf, ses, svctype, az_cred, req_soap)*

This API function decorates a request envelope with necessary ID-WSF SOAP headers and signs it, but does not send the envelope. This API is used as a building block in *tas3_call()*, which see. Usually you should use *tas3_call()* instead of this API function.

3.1.7 Requester in: status = *tas3_wsc_valid_resp(cf, ses, az_cred, res_decor_soap)*

This API function validates response envelope checking necessary ID-WSF SOAP headers and signature. This API is used as a building block in *tas3_call()*, which see. Usually you should use *tas3_call()* instead of this API function.

tas3_wsc_prepare_call() and *tas3_wsc_valid_resp()* work together as follows:

```
req_soap = tas3_wsc_prepare_call(cf, ses, svctype,
                                 url, di_opt, az_cred,
                                 "<idhrxml:Modify>...</>");
resp_soap = your_http_post_client(url, req_soap);
if (tas3_wsc_valid_resp(cf, ses, az_cred, resp_soap)) {
  xml = xml_parse(resp_soap);
  INFO("Data is " + xml->Data);
} else
  ERR("HTTP failure");
```

3.1.8 Responder in: tgtnid = *tas3_wsp_validate(cf, ses, az_cred, soap_req)*

Validate SOAP request (envelope), specified by the string `soap_req`. Service Responder should call this function to validate an inbound, received, TAS^3 request. This will

- verify signatures
- determine trust
- populate to WSP's session any credentials found in the request
- possibly perform an application independent *Responder In PEP* authorization, calling a PDP behind the scenes using *tas3_az()*.

After *tas3_wsp_validate()*, the application needs to, in application dependent way, extract from the response the application payload and process it. However, this is much simplified as there is no need to perform any further verification.

If the string `soap_req` starts by "`<e:Envelope`", then it should be a complete SOAP envelope including `<e:Header>` (and `<e:Body>`) parts.

cf TAS^3 configuration object, see *tas3_new_conf()*

ses Session object that contains the EPR cache, see *tas3_new_ses()*

az_cred (Optional) Additional authorization credentials or attributes, query string format. These credentials will be populated to the attribute pool in addition to the ones obtained from token and other sources. Then a PDP is called to get an authorization decision (matching obligations we support to those in the request, and obligations pledged by caller to those we insist on). This implements generalized (application independent) *Responder In PEP*. To implement application dependent PEP features you should call *tas3_az()* directly.

soap_req Entire SOAP envelope as a string

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

3.1.9 Responder out: soap = *tas3_wsp_decorate(cf, ses, az_cred, soap_resp)*

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

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

- add correlation headers
- possibly perform an application independent *Responder Out PEP* authorization step, calling a PDP behind the scenes using *tas3_az()*.
- apply signature

If the string starts by "`<e:Envelope`", then string should be a complete SOAP envelope including `<e:Header>` and `<e:Body>` parts. This allows caller to specify custom SOAP headers, in addition to the ones that the underlying *zxid_wsc_call()* will add. Usually the payload service will be passed as the contents of the body. If the string starts by "`<e:Body`", then the `<e:Envelope>` and `<e:Header>` are automatically added. If the string does not start by "`<e:Envelope`" or "`<e:Body`"[2], then it is assumed to be the payload content of the `<e:Body>` and the rest of the SOAP envelope is added.

cf TAS^3 configuration object, see *tas3_new_conf()*

ses Session object that contains the EPR cache

az_cred (Optional) Additional authorization credentials or attributes, query string format. These credentials will be populated to the attribute pool in addition to the ones obtained from token and other sources. Then a PDP is called to get an authorization decision (generating obligations). This implements generalized (application independent) *Responder Out PEP*. To implement application dependent PEP features you should call *tas3_az()* directly.

soap_resp XML payload as a string

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

3.1.10 Explicit Discovery: epr = *tas3_get_epr(cf, ses, svc, url, di_opt, act, n)*

N.B. This function is automatically called by *tas3_call()* so making an explicit call is seldom needed. You may consider making such call if you need to know which EPR is actually found and you want to query some properties of the EPR. You can then pass the URL, as found using *tas3_get_epr_url()*, as an argument to *tas3_call()* to constrain the call to use a specific EPR.

[2] Be careful to use the "e:" as namespace prefix if you want e:Envelope or e:Body to be detected.

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

cf TAS^3 configuration object, also used for memory allocation

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

svc Service type (usually a URN). String.

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

di_opt (Optional) Additional discovery options for selecting the service, query string format.

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

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

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

3.1.11 url = *tas3_get_epr_url(cf, epr)*

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

3.1.12 entityid = *tas3_get_epr_entid(cf, epr)*

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

3.1.13 a7n = *tas3_get_epr_a7n(cf, epr)*

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

3.1.14 SOAP Fault and Status Generation and Inspection

Error reporting using SOAP faults and TAS^3 status header is discussed in section 2.13 "Uniform Application Status and Error Reporting"

```
tas3_status* tas3_mk_tas3_status(tas3_conf* cf, const char* ctlpt, const char* sc1, const cha
tas3_fault* tas3_mk_fault(tas3_conf* cf, const char* fa, const char* fc, const char* fs, const

void tas3_set_fault(tas3_conf* cf, tas3_ses* ses, tas3_fault* flt);
tas3_fault*  tas3_get_fault(tas3_conf* cf, tas3_ses* ses);

char* tas3_get_tas3_fault_sc1(tas3_conf* cf, tas3_fault* flt);
char* tas3_get_tas3_fault_sc2(tas3_conf* cf, tas3_fault* flt);
char* tas3_get_tas3_fault_comment(tas3_conf* cf, tas3_fault* flt);
char* tas3_get_tas3_fault_ref(tas3_conf* cf, tas3_fault* flt);
char* tas3_get_tas3_fault_actor(tas3_conf* cf, tas3_fault* flt);

void tas3_set_tas3_status(tas3_conf* cf, tas3_ses* ses, tas3_status* status);
tas3_status* tas3_get_tas3_status(tas3_conf* cf, tas3_ses* ses);

char* tas3_get_tas3_status_sc1(tas3_conf* cf, tas3_status* st);
char* tas3_get_tas3_status_sc2(tas3_conf* cf, tas3_status* st);
char* tas3_get_tas3_status_comment(tas3_conf* cf, tas3_status* st);
char* tas3_get_tas3_status_ref(tas3_conf* cf, tas3_status* st);
char* tas3_get_tas3_status_ctlpt(tas3_conf* cf, tas3_status* st);
```

3.1.15 Delegated Discovery

```
void tas3_set_delegated_discovery_epr(tas3_conf* cf, tas3_ses* ses, tas3_epr* epr);
```

Allows explicit control over which Discovery Service is used, such as selecting somebody else's Discovery Service. This allows delegated access.

3.2 Java Binding

Before you start using the SSO API, you should consider using the TAS^3 SSO servlet. `tas3_sso_servlet.class` can be configured to Tomcat or other servlet container to implement SSO for payload servlets. Internally the SSO servlet calls *tas3_sso()*.

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

3.2.1 Interface and Initialization

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

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

```
import tas3java.*;
static tas3.tas3_conf cf;
static {
  System.loadLibrary("tas3jni");
  cf = tas3.new_conf_to_cf("PATH=/var/tas3/");
}
```

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

The Java binding replaces the "tas3_" prefix in function names with the class prefix "tas3.", for example *tas3_sso()* becomes *tas3.sso()* and *tas3_az()* becomes *tas3.az()*.

The TAS^3 Java interface is defined as follows

```
package tas3;

public interface tas3 {
  public static tas3_conf new_conf_to_cf(String conf);
  public static tas3_ses new_ses(tas3_conf cf);
  public static tas3_ses fetch_ses(tas3_conf cf, String sid);
  public static String sso_cf(tas3_conf cf, int qs_len, String qs,
       p_int res_len, int auto_flags);
  public static int get_ses(tas3_conf cf, tas3_ses ses, String sid);
  public static int az_cf_ses(tas3_conf cf, String qs, tas3_ses ses);
  public static int az_cf(tas3_conf cf, String qs, String sid);
  public static int az(String conf, String qs, String sid);

  public static String wsp_validate(tas3_conf cf, tas3_ses ses,
     String az_cred, String enve);
  public static String wsp_decorate(tas3_conf cf, tas3_ses ses,
     String az_cred, String enve);
  public static String call(tas3_conf cf, tas3_ses ses,
     String svctype, String url, String di_opt,
     String az_cred, String enve);
  public static tas3_epr get_epr(tas3_conf cf, tas3_ses ses,
  String svc, String url, String di_opt,
  String action, int n);
  public static String get_epr_url(tas3_conf cf, tas3_epr epr);
  public static String get_epr_entid(tas3_conf cf, tas3_epr epr);
  public static String get_epr_a7n(tas3_conf cf, tas3_epr epr);
}
```

3.2.2 Initialize: cf = *tas3.new_conf_to_cf(conf)*

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.

conf Configuration string

return Configuration object

3.2.3 New session: ses = *tas3.new_ses(cf)*

Create a new TAS3 session object. Usually a session object is created just before calling *zxidjni.wsp_validate()*.

cf Configuration object, see *tas3.new_conf_to_cf()*

return Session object

3.2.4 SSO: ret = *tas3.sso_cf_ses(cf, qs_len, qs, ses, null, auto_flags)*

cf Configuration object, see *tas3.new_conf_to_cf()*

qs_len Length of the query string. -1 = use *strlen()*

qs Query string (or POST content)

ses Session object, see *tas3.new_ses()*. Session object is modified.

res_len Result parameter. Must always pass `null` as result parameters are not supported in the Java binding.

auto_flags Automation flags

return String representing protocol action or SSO attributes

3.2.5 Authorization: decision = *tas3.az_cf_ses(cf, qs, ses)*

cf the configuration object, see *tas3.new_conf_to_cf()*

qs additional attributes that are passed to PDP

ses session object, from which most attributes come

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

3.2.6 WSC: resp_soap = *tas3.call(cf, ses, svctype, url, di_opt, az_cred, req_soap)*

cf Configuration object, see *tas3.new_conf_to_cf()*

ses Session object, used to locate EPRs, see *tas3.new_ses()*

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

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

di_opt (Optional) Additional discovery options for selecting the service, query string format

az_cred (Optional) Additional authorization credentials or attributes, query string format.

req_soap string used as SOAP body or as SOAP envelope template.

return SOAP envelope as a string

3.2.7 WSP: tgtnid = *tas3.wsp_validate(cf, ses, az_cred, soap_req)*

cf TAS^3 configuration object, see *tas3.new_conf_to_cf()*

ses Session object that contains the EPR cache, see *tas3.new_ses()*

az_cred (Optional) Additional authorization credentials or attributes, query string format.

soap_req Entire SOAP envelope as a string

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

3.2.8 WSP: soap = *tas3.wsp_decorate(cf, ses, az_cred, soap_resp)*

cf TAS^3 configuration object, see *tas3.new_conf_to_cf()*

ses Session object that contains the EPR cache

az_cred (Optional) Additional authorization credentials or attributes, query string format.

soap_resp XML payload, as a string

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

3.2.9 Explicit Discovery: epr = *tas3.get_epr(cf, ses, svc, url, di_opt, act, n)*

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

cf TAS^3 configuration object, also used for memory allocation

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

svc Service type (usually a URN)

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

di_opt (Optional) Additional discovery options for selecting the service, query string format

act (Optional) The action, or method, that must be invokable on the service

n Which matching instance is returned. 1 means first

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

3.2.10 url = *tas3.get_epr_url(cf, epr)*

cf TAS^3 configuration object, also used for memory allocation

epr An EPR object, such as obtained from *tas3_get_epr()*

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

3.2.11 entityid = *tas3.get_epr_entid(cf, epr)*

cf TAS^3 configuration object, also used for memory allocation

epr An EPR object, such as obtained from *tas3_get_epr()*

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

3.2.12 a7n = *tas3.get_epr_a7n(cf, epr)*

cf TAS^3 configuration object, also used for memory allocation

epr An EPR object, such as obtained from *tas3_get_epr()*

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

3.2.13 Available Implementations (Non-normative)

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

3.3 PHP Binding

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

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

3.3.1 Application Level Integration

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

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

3.3.2 cf = *tas3_new_conf_to_cf(conf)*

conf Configuration string

return Configuration object

3.3.3 ses = *tas3_new_ses(cf)*

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

cf Configuration object

return Session object

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

cf Configuration object, see *tas3_new_conf_to_cf()*

qs_len Length of the query string. -1 = use *strlen()*

qs Query string (or POST content)

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

res_len Should always be passed as null (result parameter is not supported for PHP).

auto_flags Automation flags

return String representing protocol action or SSO attributes

Example

```
<?
$qs = $_SERVER['REQUEST_METHOD'] == 'GET'
      ? $_SERVER['QUERY_STRING']
      : file_get_contents('php://input');
$ses = tas3_new_ses($cf);
```

```
$res = tas3_sso_cf_ses($cf, -1, $qs, $ses, null, 0x1814);
switch (substr($res, 0, 1)) {
case 'L': header($res); exit;  # Redirect (Location header)
case '<': header('Content-type: text/xml'); echo $res; exit;
case 'n': exit;   # Already handled
case 'e': my_render_idp_select();
case 'd': break;  # Logged in case
default:  die("Unknown res($res)");
}

if (tas3_az_cf_ses($cf, "Action=Show", $ses)) {
    echo "Permit.\n";
    # Render protected content here
} else {
    echo "<b>Deny.</b>";
}
?>
```

3.3.5 Authorization: decision = *tas3_az_cf_ses(cf, qs, ses)*

cf the configuration object

qs additional attributes that are passed to PDP

ses session object, from which most attributes come

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

3.3.6 WSC: resp_soap = *tas3_call(cf, ses, svctype, url, di_opt, az_cred, req_soap)*

cf Configuration object, see *tas3_new_conf_to_cf()*

ses Session object, used to locate EPRs, see *tas3_new_ses()*

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

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

di_opt (Optional) Additional discovery options for selecting the service, query string format

az_cred (Optional) Additional authorization credentials or attributes, query string format.

req_soap string used as SOAP body or as SOAP envelope template.

return SOAP envelope as a string

Example

```
$ret = tas3_call($cf, $ses, "urn:id-sis-idhrxml:2007-06:dst-2.1",
                 null, null, null,
                 "<idhrxml:Query>" .
                   "<idhrxml:QueryItem>" .
                     "<idhrxml:Select>$criteria</idhrxml:Select>" .
                   "</idhrxml:QueryItem>" .
                 "</idhrxml:Query>");
```

3.3.7 WSP: tgtnid = *tas3_wsp_validate(cf, ses, az_cred, soap_req)*

cf TAS^3 configuration object, see *tas3_new_conf()*

ses Session object that contains the EPR cache, see *tas3_new_ses()*

az_cred (Optional) Additional authorization credentials or attributes, query string format.

soap_req Entire SOAP envelope as a string

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

3.3.8 WSP: soap = *tas3_wsp_decorate(cf, ses, az_cred, soap_resp)*

cf TAS^3 configuration object, see *tas3_new_conf()*

ses Session object that contains the EPR cache

az_cred (Optional) Additional authorization credentials or attributes, query string format.

soap_resp XML payload, as a string

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

3.3.9 Explicit Discovery: epr = *tas3_get_epr(cf, ses, svc, url, di_opt, act, n)*

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

cf TAS^3 configuration object, also used for memory allocation

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

svc Service type (usually a URN)

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

di_opt (Optional) Additional discovery options for selecting the service, query string format

act (Optional) The action, or method, that must be invokable on the service

n Which matching instance is returned. 1 means first

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

3.3.10 url = *tas3_get_epr_url(cf, epr)*

cf TAS^3 configuration object, also used for memory allocation

epr An EPR object, such as obtained from *tas3_get_epr()*

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

3.3.11 entityid = *tas3_get_epr_entid(cf, epr)*

cf TAS^3 configuration object, also used for memory allocation

epr An EPR object, such as obtained from *tas3_get_epr()*

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

3.3.12 a7n = *tas3_get_epr_a7n(cf, epr)*

cf TAS^3 configuration object, also used for memory allocation

epr An EPR object, such as obtained from *tas3_get_epr()*

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

3.3.13 Available Implementations (Non-normative)

This binding is implemented by php_zxid module, available as part of the `zxid.org`

3.4 C and C++ Binding

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

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

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

3.4.1 cf = *tas3_new_conf_to_cf(conf)*

Prototype

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

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.

conf Configuration string

return Configuration object

3.4.2 ses = *tas3_new_ses(cf)*

Prototype

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

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

cf Configuration object

return Session object

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

Prototype

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

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

cf Configuration object, see *tas3_new_conf_to_cf()*

qs_len Length of the query string. -1 = use *strlen()*

qs Query string (or POST content)

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

res_len Result parameter. If non-null, will be set to the length of the returned string

auto_flags Automation flags

return String representing protocol action or SSO attributes

Example

```
{
  tas3_conf* cf  = tas3_new_conf_to_cf("PATH=/var/tas3/");
  tas3_ses*  ses = tas3_new_ses(cf);
  char* ret = tas3_sso_cf_ses(cf, -1, env("QUERY_STRING"), ses, 0, 0x1800);
  switch (ret[0]) {
  case ’d’: break;  /* Successful login */
  ...               /* Processing other outcomes omitted for brevity. */
  }
  if (tas3_az_cf_ses(cf, "", ses)) {
    /* SSO successful and authorization permit. Do some work. */
  } else {
    /* SSO successful but authorization denied */
  }
}
```

3.4.4 Authorization: decision = *tas3_az_cf_ses(cf, qs, ses)*

Prototype

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

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

cf the configuration object

qs additional attributes that are passed to PDP

ses session object, from which most attributes come

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

3.4.5 WSC: resp_soap = *tas3_call(cf, ses, svctype, url, di_opt, az_cred, req_soap)*

Prototype

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

cf Configuration object, see *tas3_new_conf_to_cf()*

ses Session object, used to locate EPRs, see *tas3_new_ses()*

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

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

di_opt (Optional) Additional discovery options for selecting the service, query string format

az_cred (Optional) Additional authorization credentials or attributes, query string format.

req_soap string used as SOAP body or as SOAP envelope template.

return SOAP envelope as a string

3.4.6 resp_soap = tas3_callf(cf, ses, svctype, url, di_opt, az_cred, fmt, ...)

Prototype

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

The *tas3_callf()* variant, which allows *printf(3)* style formatting, is highly convenient for C programmers. Others will probably use the plan *tas3_call()* and rely on language's native abilities to construct the string.

cf Configuration object, see *tas3_new_conf_to_cf()*

ses Session object, used to locate EPRs, see *tas3_new_ses()*

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

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

di_opt (Optional) Additional discovery options for selecting the service, query string format

az_cred (Optional) Additional authorization credentials or attributes, query string format.

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

return SOAP envelope as a string

3.4.7 WSP: tgtnid = *tas3_wsp_validate(cf, ses, az_cred, soap_req)*

Prototype

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

cf TAS^3 configuration object, see *tas3_new_conf()*

ses Session object that contains the EPR cache, see *tas3_new_ses()*

az_cred (Optional) Additional authorization credentials or attributes, query string format.

soap_req Entire SOAP envelope as a string

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

3.4.8 WSP: soap = *tas3_wsp_decorate(cf, ses, az_cred, soap_resp)*

Prototype

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

cf TAS^3 configuration object, see *tas3_new_conf()*

ses Session object that contains the EPR cache

az_cred (Optional) Additional authorization credentials or attributes, query string format.

soap_resp XML payload as a string

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

3.4.9 WSP: soap = tas3_wsp_decoratef(cf, ses, az_cred, fmt, ...)

Prototype

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

cf TAS^3 configuration object, see *tas3_new_conf()*

ses Session object that contains the EPR cache

az_cred (Optional) Additional authorization credentials or attributes, query string format.

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

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

3.4.10 Explicit Discovery: epr = *tas3_get_epr(cf, ses, svc, url, di_opt, act, n)*

Prototype

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

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

cf TAS^3 configuration object, also used for memory allocation

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

svc Service type (usually a URN)

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

di_opt (Optional) Additional discovery options for selecting the service, query string format

act (Optional) The action, or method, that must be invokable on the service

n Which matching instance is returned. 1 means first

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

3.4.11 url = *tas3_get_epr_url(cf, epr)*

Prototype

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

cf TAS^3 configuration object, also used for memory allocation

epr An EPR object, such as obtained from *tas3_get_epr()*

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

3.4.12 entityid = *tas3_get_epr_entid(cf, epr)*

Prototype

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

cf TAS^3 configuration object, also used for memory allocation

epr An EPR object, such as obtained from *tas3_get_epr()*

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

3.4.13 a7n = *tas3_get_epr_a7n(cf, epr)*

Prototype

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

cf TAS^3 configuration object, also used for memory allocation

epr An EPR object, such as obtained from *tas3_get_epr()*

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

3.4.14 Available Implementations (Non-normative)

This binding is implemented, at least, by `zxid.org` open source implementation, which serves as the reference implementation of the TAS^3 core security architecture.

N.B. The *tas3_sso()* API is implemented by zxid's *zxid_simple()* API.

3.5 Other Language Bindings

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

- C# / .Net / Mono
- Perl (currently `zxid.org` derived Net::SAML perl module, available from `cpan.org`, supports most functionality of TAS^3 API, but this is unofficial)
- Python
- Ruby

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

4 Deployment and Integration Models (Non-normative)

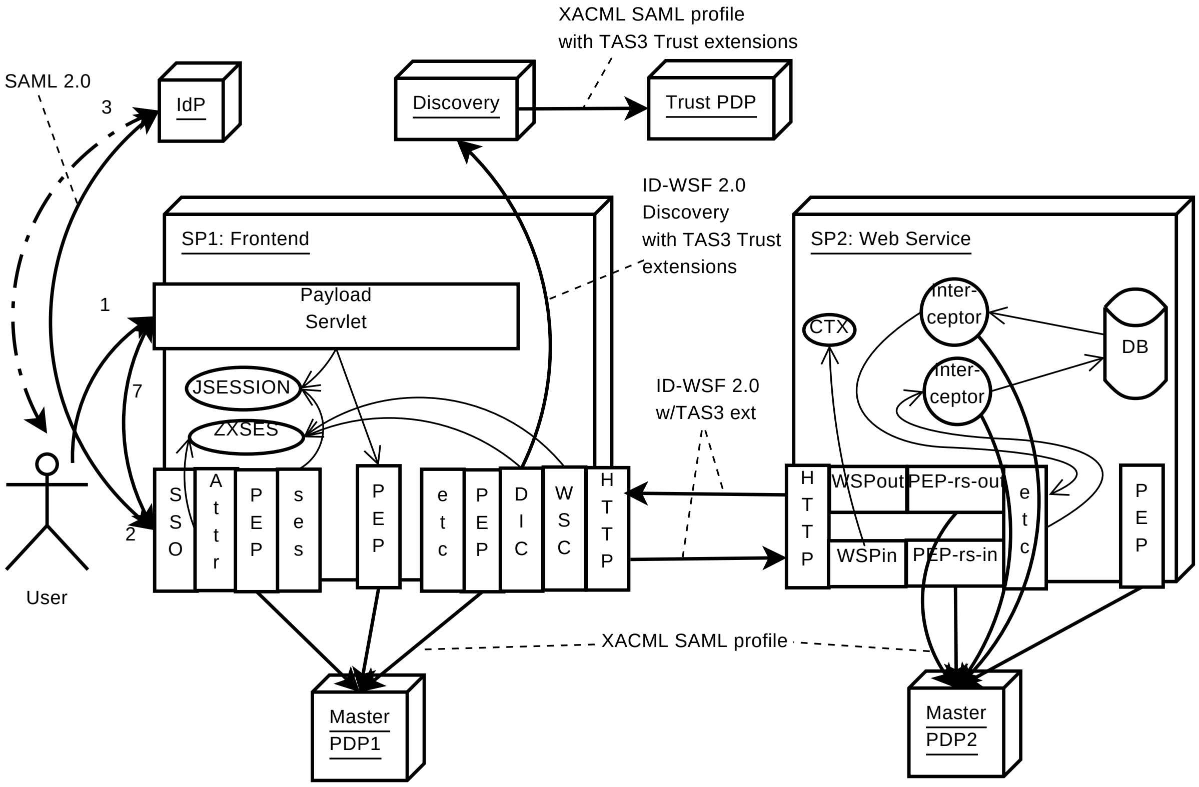


Figure 4.1: A deployment architecture for SSO and web service call.

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

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

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

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.

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.

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^3 certified, APIs that make the instrumenting job easy.

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

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

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

2. Perform SSO (SP side)
3. Perform SSO, IdP side including authenticating user and shipping attributes
4. Gater additional attributes, if needed ("Attr")
5. Authorize access to FE (PEP-Rs-In of FE) ("PEP")
6. Populate session of the payload servlet ("ses")
7. Redirect user to protected resource he was trying to access on the protected resource.
8. Application dependent PEP calls PDP if needed. ("PEP")
9. Call web service, including
   a. Application dependent processing steps ("etc")
   b. Authorize the call (PEP-Rq-Out) ("PEP")
   c. Discover suitable service, performing Trust and Privacy Negotiation (may need interaction at frontend web gui) if needed. ("DIC")
   d. Decorate request with TAS3 specific SOAP headers and sign. ("WSC")
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.

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.[1] The TAS^3 binary modules for SSO are generically called T3-SSO-*.

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^3 can be hooked in without interference[2]).

The API realization of WSC is a function, *tas3_call()* (see TAS^3 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.

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.[3]

4.1.1 Integration Using ZXID (Non-normative)

Further information about using ZXID for TAS^3 is available in README.zxid-tas3, `zxid-tas3.pd`, and `zxid-java.pd`

The official TAS^3 API is provided by `tas3.h` which maps the TAS^3 API definitions to the underlying zxid ones.

The Java realization of SSO is provided by zxidsrvlet class and servlet. This is packaged as TAS^3 binary module T3-SSO-ZXID-JAVA.

The web server realization of SSO is provided by mod_auth_saml Apache module (`mod_auth_saml.so`). It is packaged as TAS^3 binary module T3-SSO-ZXID-MODAUTHSAML.

[1] In mod_auth_saml realization even step (1) can be accomplished by configuring the web server.

[2] 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>`).

[3] In ZXID realization the HTTP client is libcurl from curl.haxx.se

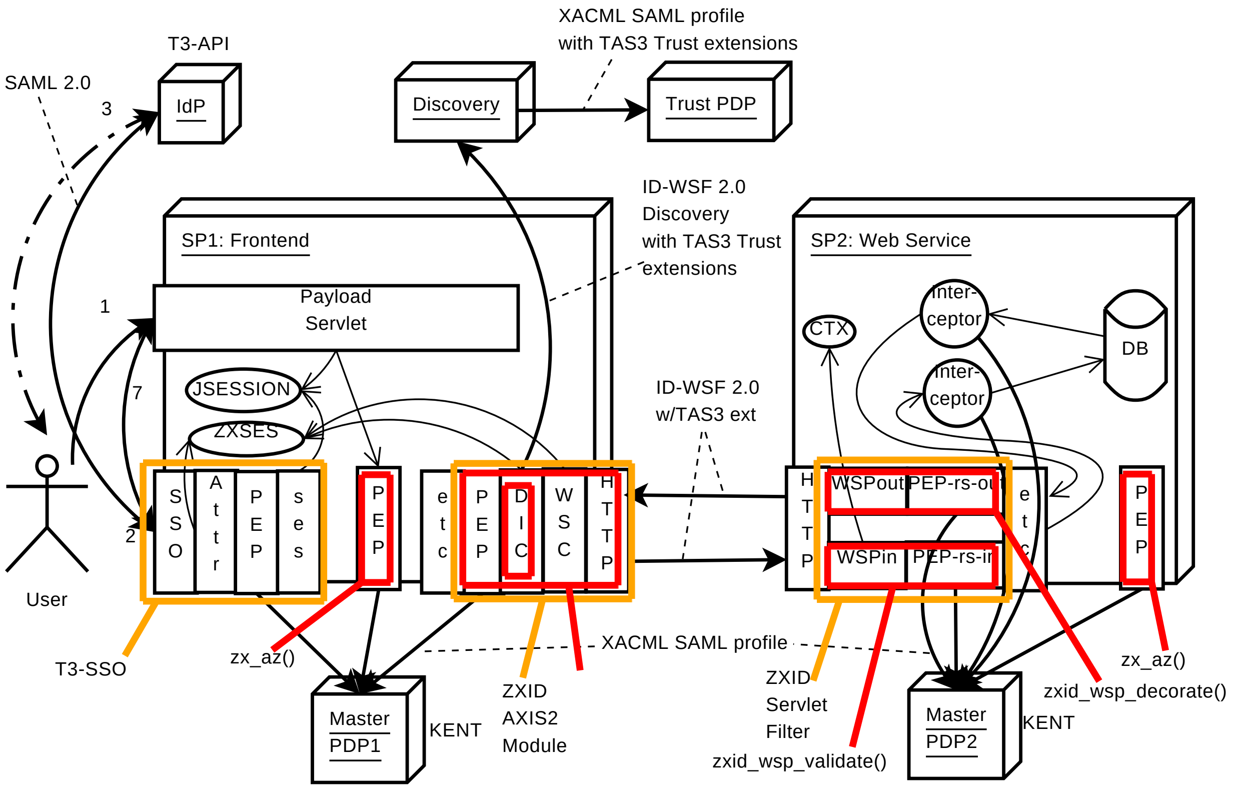


Figure 4.2: API and modules for SSO and web service call.

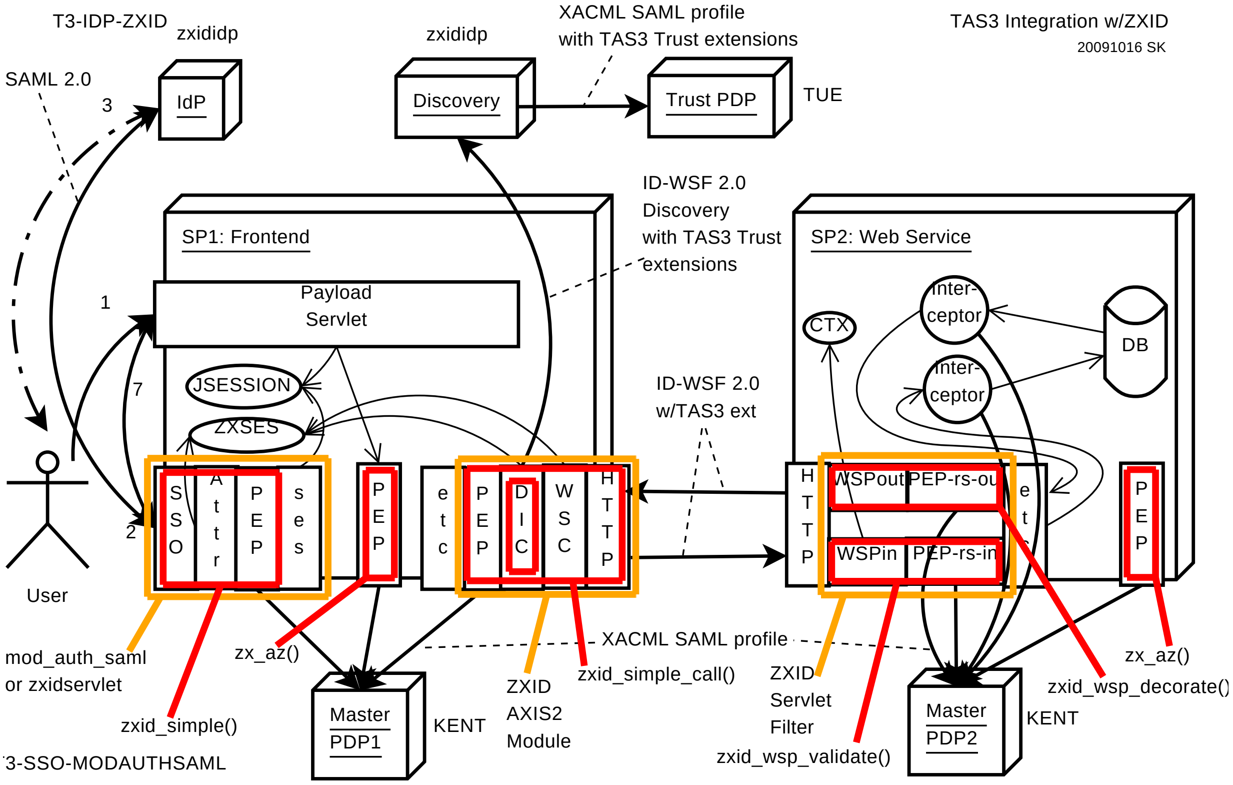


Figure 4.3: ZXID specific API and modules for SSO and web service call.

2167 API realization of SSO is provided by *zxid_simple()* in `libzxid.a`. This is packaged as TAS^3 binary
2168 module T3-SSO-ZXID-PHP.[4] Other language binding specific modules are expected in the future.

[4] Although not TAS3 packaged, Net::SAML perl module provides the same functionality.

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

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

4.2 Web Services Provider Integration Model (Non-normative)

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

A. Listen for HTTP requests (typically done by platform)

B. Parse and validate a web services request, e.g. call *tas3_wsp_validate()*. This involves checking for valid signature from trusted authority.

C. Authorize the request, extracting from the request the pledges (in `<b:UsageDirective>`) ("PEP-Rs-In").

D. Apply other filters and post processing steps ("etc")

E. Authorize each data item separately using input interceptor. For queries this is usually a no-op, but for creates or updates this is meaningful. When data is accepted for the repository, the authorization step can result in obligations or sticky-policies being written into the database along side the data itself.

   The authorization is configurable according to Application Independent PEP configuration, described elsewhere, or Application Dependent PEP approach can be taken, calling the PDP directly ("PEP").

F. Authorize each returned data item separately using input interceptor. Usually applicable to query results. The per item authorization will apply systemwide and item specific policies (sticky policies) and obligations and produce a deny or permit-with-obligations response.

   The authorization is configurable according to Application Independent PEP configuration, described elsewhere, or Application Dependent PEP approach can be taken, calling the PDP directly ("PEP").

G. Authorize the response in aggregate ("PEP-Rs-Out"). At this stage one of the most important verifications is to compare the pledges collected in step C ("PEP-Rs-In") and filter out any data whose obligations are stricter.

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

H. Decorate the response with TAS3 specific SOAP headers. This is typically done by calling *tas3_wsp_decorate()*.

I. Send the response. This is typically done by platform dependent means.

5 Resilient Deployment Architecture (Non-normative)

This section addresses Req. *D1.2-2.8-Avail*.

For TAS^3 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.

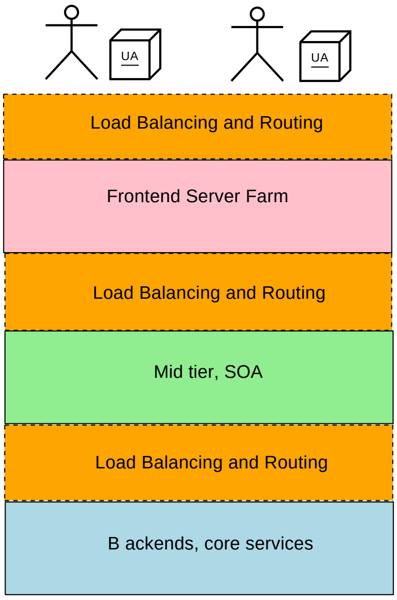
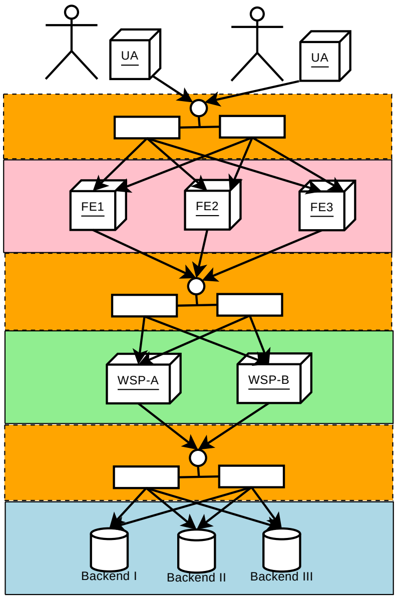

UA
UA
Load Balancing and Routing
Frontend Server Farm
Load Balancing and Routing
Mid tier, SOA
Load Balancing and Routing
B ackends, core services

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



Virtual IP
Redundancy using VRRP
Hardware Loadbalancers

Figure 5.2: Resiliency implemented using hardware load balancers.

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) [**?**].

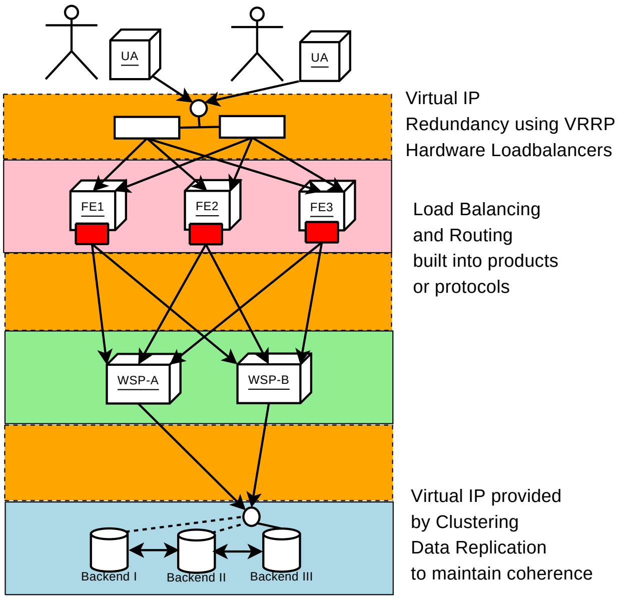


Figure 5.3: Resiliency implemented using software load-balancing-fail-over functionality and clustering.

5.1 Zero Downtime Updates

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

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

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

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

6 Feasibility and Performance Analysis (Non-normative)

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

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^3 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.

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 neglible compared to TCP + TLS handshake and digital signatures, we will not consider communication cost in our calculations.

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.

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.

Z Authorization cost. Evaluation of rule set will depend heavily on the particular ruleset and its implementation technology. Some rulesets are know 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).

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^3 Architecture, the payload is assumed to be free.

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

The cost is unevenly divided among the entities in the TAS^3 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.

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).

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.

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.

6.1 Single use of single web service

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.

Table 6.1: Cost of TAS^3 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=28.5	7T+11S+19X=40.5	2T+6S+11X+3Z=21.5	2T+6S+11X=20	2T+5S+11X+2Z=20	12T+18S+54X=90	4T+36X+18Z=51

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

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.

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

6.1.1 Cost without auditing

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

Table 6.3: Cost of TAS^3 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

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

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

6.1.2 Cost without auditing and without authorization

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

Table 6.4: Cost of TAS^3 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

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

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

6.1.3 Cost without XML

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

Table 6.5: Cost of TAS^3 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
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

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

6.2 Session of 3 frontends and five web services

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

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

Table 6.6: Cost of TAS^3 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
29. Process Obli		2t+S+2X=3		2t+S+2X=3		2(2t+S+3X)=8	2(t+2X+Z)=5

Table 6.6 (continued): Cost of TAS^3 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

Table 6.6 (continued): Cost of TAS^3 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	36T+176S+528X	T+352X+176Z
TOTAL RSA	=92	=305	=220.5	=273	=255	=758	=443

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.

7 Best Practises

1. Each entity chooses its own Entity ID. When you are setting up a SP, you choose your Entity ID and the *IdP(s)* MUST be able to adapt to your choice. Similarily, an IdP decides its own Entity ID and all SPs MUST be able to adapt to it.

2. Entity IDs MUST be unique within a Circle of Trust (CoT). Given that CoT relationships may change from time to time, its best to choose Entity ID so that it is globally unique. If Entity ID contains a domain name as a component, then the *globally unique* property tends to be enforced by the domain name allocation system.

3. Entity ID SHOULD be the Well Known Location (WKL), i.e. the URL from which the metadata can be fetched.

4. Providing metadata by URL, ideally by the Entity ID, SHOULD always be enabled. This greatly facilitates configuration.

5. After you get an installation to work, be sure to review whether the default configuration is appropriate for production use

   a. Decide whether you want to run open federation and disable it if needed.

   b. Prune your Circle of Trust. List who you trust and delete the misfits.

   c. Check validity time tolerances you accept. The defaults may be rather generous for production use.

   d. Review that you did not turn off any signature validation just to get it to work. All signature validations are there for reason and you should not go to production if any of them fail.

   e. Check permissions on private keys and think whether your private keys, including web server SSL one, are protected. Could they have been compromised during trial period?

   f. Check that your public image is conveyed right in your metadata. Orgqanization name, contact URLs, logotype, etc. However, be forewarned that changing these on last minute changes your metadata and you may need to engage in an additional round of metadata exchanges when you go to production.

   g. Make sure you have a solution in place to keep your audit trail in case you ever have to go to court. See `zxid-log.pd` for details. You may also want to think about encrypting or deleting some items after a while to reduce your liability for breaches.

8 Annex A: Examples

These XML blobs, taken from [**?**], 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.

8.1 SAML 2.0 Artifact Response with SAML 2.0 SSO Assertion and Two Bootstraps

Both bootstraps illustrate SAML assertion as bearer token.

```
<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="RCCzu13z77SiSXqsFp1u1"
         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>
```

```
    <ds:CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
    <ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
    <ds:Reference URI="#ASSE6bgfaV-sapQsAilXOvBu">
      <ds:Transforms>
        <ds:Transform Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature
        <ds:Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/></>
      <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
      <ds:DigestValue>r8OvtNmq5LkYwCNg6bsRZAdT4NE=</></></>
  <ds:SignatureValue>GtWVZzHYW54ioHk/C7zjDRThohrpwC4=</></>

<sa:Subject>
  <sa:NameID
      Format="urn:oasis:names:tc:SAML:2.0:nameid-format:persistent"
      NameQualifier="https://a-idp.liberty-iop.org:8881/idp.xml">PB5fLIA4lRU2bH4HkQsn
  <sa:SubjectConfirmation
      Method="urn:oasis:names:tc:SAML:2.0:cm:bearer">
    <sa:SubjectConfirmationData
        NotOnOrAfter="2007-02-10T06:37:41Z"
        Recipient="https://sp1.zxidsp.org:8443/zxidhlo?o=B"/></></>

<sa:Conditions
    NotBefore="2007-02-10T05:32:42Z"
    NotOnOrAfter="2007-02-10T06:37:42Z">
  <sa:AudienceRestriction>
    <sa:Audience>https://sp1.zxidsp.org:8443/zxidhlo?o=B</></></>

<sa:Advice>

  <!-- This assertion is the credential for the ID-WSF 1.1 bootstrap (below). -->

  <sa:Assertion
      ID="CREDOTGAkvhNoP1aiTq4bXBg"
      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>
        <ds:CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#
        <ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
        <ds:Reference URI="#CREDOTGAkvhNoP1aiTq4bXBg">
          <ds:Transforms>
            <ds:Transform Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signa
            <ds:Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/></>
          <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
          <ds:DigestValue>dqq/28hw5eEv+ceFyiLImeJ1P8w=</></></>
      <ds:SignatureValue>UKlEgHKQwuoCE=</></>
    <sa:Subject>
      <sa:NameID/>  <!-- *** Bug here!!! -->
      <sa:SubjectConfirmation
          Method="urn:oasis:names:tc:SAML:2.0:cm:bearer"/></>
    <sa:Conditions
    NotBefore="2007-02-10T05:32:42Z"
    NotOnOrAfter="2007-02-10T06:37:42Z">
   <sa:AudienceRestriction>
    <sa:Audience>https://sp1.zxidsp.org:8443/zxidhlo?o=B</></></></></>

<sa:AuthnStatement
  AuthnInstant="2007-02-10T05:37:42Z"
  SessionIndex="1171085858-4">
 <sa:AuthnContext>
  <sa:AuthnContextClassRef>
   urn:oasis:names:tc:SAML:2.0:ac:classes:Password</></></>

<sa:AttributeStatement>

 <!-- Regular attribute -->

 <sa:Attribute
   Name="cn"
   NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:basic">
  <sa:AttributeValue>Sue</></>

 <!-- ID-WSF 1.1 Bootstrap for discovery. See also the Advice, above. -->

 <sa:Attribute
   Name="DiscoveryResourceOffering"
   NameFormat="urn:liberty:disco:2003-08">
  <sa:AttributeValue>
   <di12:ResourceOffering
     xmlns:di12="urn:liberty:disco:2003-08"
     entryID="2">
    <di12:ResourceID>
     https://a-idp.liberty-iop.org/profiles/WSF1.1/RID-DISCO-sue</>
    <di12:ServiceInstance>
     <di12:ServiceType>urn:liberty:disco:2003-08</>
     <di12:ProviderID>https://a-idp.liberty-iop.org:8881/idp.xml</>
     <di12:Description>
      <di12:SecurityMechID>urn:liberty:security:2005-02:TLS:Bearer</>
      <di12:CredentialRef>CREDOTGAkvhNoP1aiTq4bXBg</>
      <di12:Endpoint>https://a-idp.liberty-iop.org:8881/DISCO-S</></></>
    <di12:Abstract>Symlabs Discovery Service Team G</></></></>

 <!-- ID-WSF 2.0 Bootstrap for Discovery. The credential (bearer token) is inline. -

 <sa:Attribute
   Name="urn:liberty:disco:2006-08:DiscoveryEPR"
   NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:uri">
  <sa:AttributeValue>
   <wsa:EndpointReference
     xmlns:wsa="http://www.w3.org/2005/08/addressing"
     xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecuri
     notOnOrAfter="2007-02-10T07:37:42Z"
     wsu:Id="EPRIDcjP8ObO9In47SDjO9b37">
    <wsa:Address>https://a-idp.liberty-iop.org:8881/DISCO-S</>
```

```
<wsa:Metadata xmlns:di="urn:liberty:disco:2006-08">
  <di:Abstract>SYMfiam Discovery Service</>
  <sbf:Framework xmlns:sbf="urn:liberty:sb" version="2.0"/>
  <di:ProviderID>https://a-idp.liberty-iop.org:8881/idp.xml</>
  <di:ServiceType>urn:liberty:disco:2006-08</>
  <di:SecurityContext>
    <di:SecurityMechID>urn:liberty:security:2005-02:TLS:Bearer</>

    <sec:Token
        xmlns:sec="urn:liberty:security:2006-08"
        usage="urn:liberty:security:tokenusage:2006-08:SecurityToken">

      <sa:Assertion
          ID="CREDV6ZBMyicmyvDq9pLIoSR"
          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>
            <ds:CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/x
            <ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#
            <ds:Reference URI="#CREDV6ZBMyicmyvDq9pLIoSR">
              <ds:Transforms>
                <ds:Transform Algorithm="http://www.w3.org/2000/09/xmldsig#en
                <ds:Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c1
              <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#s
              <ds:DigestValue>o2SgbuKIBzl4e0dQoTwiyqXr/8Y=</></></>
          <ds:SignatureValue>hHdUKaZ//cZ8UYJxvTReNU=</></>
        <sa:Subject>
          <sa:NameID
              Format="urn:oasis:names:tc:SAML:2.0:nameid-format:persistent"
              NameQualifier="https://a-idp.liberty-iop.org:8881/idp.xml">
            9my93VkP3tSxEOIb3ckvjLpn0pa6aV3yFXioWX-TzZI=</>
          <sa:SubjectConfirmation
              Method="urn:oasis:names:tc:SAML:2.0:cm:bearer"/></>
        <sa:Conditions
            NotBefore="2007-02-10T05:32:42Z"
            NotOnOrAfter="2007-02-10T06:37:42Z">
          <sa:AudienceRestriction>
            <sa:Audience>https://a-idp.liberty-iop.org:8881/idp.xml</></></>
        <sa:AuthnStatement AuthnInstant="2007-02-10T05:37:42Z">
          <sa:AuthnContext>
            <sa:AuthnContextClassRef>
              urn:oasis:names:tc:SAML:2.0:ac:classes:Password</></></></></></><
```

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

8.2 ID-WSF 2.0 Call with X509v3 Sec Mech

```
<e:Envelope
```

```
   xmlns:e="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:b="urn:liberty:sb:2005-11"
   xmlns:sec="urn:liberty:security:2005-11"
   xmlns:wsse="http://docs.oasis-open.org/wss/20 04/01/oasis-200401-wss-wssecurity-secext-1.0.
   xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.x
   xmlns:wsa="http://www.w3.org/2005/08/ addressing">
 <e:Header>
   <wsa:MessageID wsu:Id="MID">123</>
   <wsa:To wsu:Id="TO">...</>
   <wsa:Action wsu:Id="ACT">urn:xx:Query</>
   <wsse:Security mustUnderstand="1">
     <wsu:Timestamp wsu:Id="TS"><wsu:Created>2005-06-17T04:49:17Z</></>
     <wsse:BinarySecurityToken
         ValueType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-
         wsu:Id="X509Token"
         EncodingType="http://docs.oas is-open.org/wss/2004/01/oasis-200401-wss-soap-message-se
       MIIB9zCCAWSgAwIBAgIQ...</>
     <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
       <ds:SignedInfo>
         <ds:Reference URI="#MID">...</>
         <ds:Reference URI="#TO">...</>
         <ds:Reference URI="#ACT">...</>
         <ds:Reference URI="#TS">...</>
         <ds:Reference URI="#X509">
           <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
           <ds:DigestValue>Ru4cAfeBAB</></>
         <ds:Reference URI="#BDY">
           <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
           <ds:DigestValue>YgGfS0pi56p</></></>
       <ds:KeyInfo><wsse:SecurityTokenReference><wsse:Reference URI="#X509"/></></>
       <ds:SignatureValue>HJJWbvqW9E84vJVQkjDElgscSXZ5Ekw==</></></></>
 <e:Body wsu:Id="BDY">
   <xx:Query/></></>
```

The salient features of the above XML blob are

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

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

8.3 ID-WSF 2.0 Call with Bearer (Binary) Sec Mech

```
<e:Envelope
   xmlns:e="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:b="urn:liberty:sb:2005-11"
   xmlns:sec="urn:liberty:security:2005-11"
   xmlns:wsse="http://docs.oasis-open.org/wss/20 04/01/oasis-200401-wss-wssecurity-secext-1.0.
   xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.x
   xmlns:wsa="http://www.w3.org/2005/03/ addressing">
 <e:Header>
   <wsa:MessageID wsu:Id="MID">...</>
   <wsa:To wsu:Id="TO">...</>
   <wsa:Action wsu:Id="ACT">urn:xx:Query</>
   <wsse:Security mustUnderstand="1">
     <wsu:Timestamp wsu:Id="TS">
       <wsu:Created>2005-06-17T04:49:17Z</></>
     <wsse:BinarySecurityToken
         ValueType="anyNSPrefix:ServiceSess ionContext"
         EncodingType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-sec
         wsu:Id="BST">
       mQEMAzRniWkAAAEH9RWir0eKDkyFAB7PoFazx3ftp0vWwbbzqXdgcX8fpEqSr1v4
       YqUc7OMiJcBtKBp3+jlD4HPUaurIqHA0vrdmMpM+sF2BnpND118f/mXCv3XbWhiL
       VT4r9ytfpXBluelOV93X8RUz4ecZcDm9e+IEG+pQjnvgrSgac1NrW5K/CJEOUUjh
       oGTrym0Ziutezhrw/gOeLVtkywsMgDr77gWZxRvw01w1ogtUdTceuRBIDANj+KVZ
       vLKlTCaGAUNIjkiDDgti=</>
     <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig #">
       <ds:SignedInfo>
         <ds:Reference URI="#MID">...</>
         <ds:Reference URI="#TO">...</>
         <ds:Reference URI="#ACT">...</>
         <ds:Reference URI="#TS">...</>
         <ds:Reference URI="#BST">...</>
         <ds:Reference URI="#BDY">
           <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1 "/>
           <ds:DigestValue>YgGfS0pi56pu</></></>
       ...</></></>
 <e:Body wsu:Id="BDY">
   <xx:Query/></></>
```

8.4 ID-WSF 2.0 Call with Bearer (SAML) Sec Mech

```
<e:Envelope
   xmlns:e="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:sb="urn:liberty:sb:2005-11"
   xmlns:sec="urn:liberty:security:2005-11"
   xmlns:wsse="http://docs.oasis-open.org/wss/20 04/01/oasis-200401-wss-wssecurity-secext-1.0.x
   xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.x
   xmlns:wsa="http://www.w3.org/2005/08/addressing"
   xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
   xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
 <e:Header>
   <sbf:Framework version="2.0-simple" e:mustUnderstand="1"
     e:actor="http://schemas.../next"
     wsu:Id="SBF"/>
   <wsa:MessageID wsu:Id="MID">...</>
   <wsa:To wsu:Id="TO">...</>
   <wsa:Action wsu:Id="ACT">urn:xx:Query</>
   <wsse:Security mustUnderstand="1">
     <wsu:Timestamp wsu:Id="TS">
       <wsu:Created>2005-06-17T04:49:17Z</></>
```

```
    <sa:Assertion
        xmlns:sa="urn:oasis:names:tc:SAML:2.0:assertion"
        Version="2.0"
        ID="A7N123"
        IssueInstant="2005-04-01T16:58:33.173Z">
      <sa:Issuer>http://idp.symdemo.com/idp.xml</>
      <ds:Signature>...</>
      <sa:Subject>
        <sa:EncryptedID>
          <xenc:EncryptedData>U2XTCNvRX7Bl1NK182nmY00TEk==</>
          <xenc:EncryptedKey>...</></>
        <sa:SubjectConfirmation Method="urn:oasis:names:tc:SAML:2.0:cm:bearer"/></>
      <sa:Conditions
          NotBefore="2005-04-01T16:57:20Z"
          NotOnOrAfter="2005-04-01T21:42:4 3Z">
        <sa:AudienceRestrictionCondition>
          <sa:Audience>http://wsp.zxidsp.org</></></>
      <sa:AuthnStatement
          AuthnInstant="2005-04-01T16:57:30.000Z"
          SessionIndex="6345789">
        <sa:AuthnContext>
          <sa:AuthnContextClassRef>
            urn:oasis:names:tc:SAML:2.0:ac:classes:PasswordProtectedTransport</></></>
      <sa:AttributeStatement>
        <sa:EncryptedAttribute>
          <xenc:EncryptedData Type="http://www.w3.org/2001/04/xmlenc#Element">
            mQEMAzRniWkAAAEH9RbzqXdgcX8fpEqSr1v4=</>
          <xenc:EncryptedKey>...</></></></></>

    <wsse:SecurityTokenReference
        xmlns:wsse11="..."
        wsu:Id="STR1"
        wsse11:TokenType="http://docs.oasis-open.org/wss/oasis-wss-saml-token-profile-1.1#SAM
      <wsse:KeyIdentifier
          ValueType="http://docs.oasis-open.org/wss/oasis-wss-saml-token-profile-1.1#SAMLID">
        A7N123</></>

    <ds:Signature>
      <ds:SignedInfo>
        <ds:Reference URI="#MID">...</>
        <ds:Reference URI="#TO">...</>
        <ds:Reference URI="#ACT">...</>
        <ds:Reference URI="#TS">...</>
        <ds:Reference URI="#STR1">
          <ds:Transform Algorithm="...#STR-Transform">
            <wsse:TransformationParameters>
              <ds:CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20
        <ds:Reference URI="#BDY"/></>
      ...</></></>
  <e:Body wsu:Id="BDY">
    <xx:Query/></></>
```

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

9 Annex B: Technical Self Assessment Questionnaire

This questionnaire is to be used in partner intake process of a TAS^3 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 [**?**], 11.6 Annex IV "Self Assessment Questionnaire".

9.1 Overview and Scope

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

   Installation Name: ____________________________

2. Please supply your organizational and contact details

   ____

   ____

   ____

   Technical contact for clarifications: ____________

   Who filled this questionnaire: _____________

   Date when filled or amended: ________

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

   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).

   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)

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

   d. (__) Identity Aggregator or Linking Service

   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)

   f. (__) Trust and Reputation provider towards external parties

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

   h. (__) Online Compliance Testing Provider

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

   j. (__) Other, please specify: __________________________________

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".

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

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

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

Table 9.1: 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.				

a. (__) Complete outsource to a partner, which: ____________________

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

b. (__) Software as a Service (SaaS), operated by you.
   Which software or partner: _____________________, version: ___
   Your SaaS provider should help you answer the technical questions.

c. (__) Operate commercial software on servers administered by you (e.g. own server, hosted root server, server on Amazon Elastic Cloud, etc.)
   Which software: _____________________, version: ___

d. (__) Operate open source software on servers administered by you (e.g. own server, hosted root server, server on Amazon Elastic Cloud, etc.)
   Which software: _____________________, version: ___

e. (__) Operate software developed by you or for you
   Which software: _____________________, version: ___

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

   Number of potential users: _______________

   Number of regular or frequent users: _______________

   Number of tasks performed by a regular user on typical working day on your service: _______________

   Any performance targets you expect from the system, such as maximum latency or required throughput: ______________________

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

9.2 System Entity Credentials and Private Keys

In TAS^3, services and other system entities are identified using X509 digital certificates. They are used in TLS connections for authentication using Client TLS and they are used for digital signatures.

Responsible management of the private keys associated with the digital certificates is the corner stone of TAS^3 accountability and liability framework. Your organization will be held responsible for all actions performed using your private keys.

1. Which certification authority do you use for issuance of certificates? (if selfissued, indicate who in your organization is responsible)

   ____

2. How do you generate private key and certification request?

   ____

3. What measures are in place to ensure that the private key remains confidential during generation, certificate issuance, and installation process? How do you know that no copy is left on any device (e.g. USB stick of a consultant) used to handle the private key?

   ____

4. What backup arrangements do you have for the private key and how are they kept confidential?

   ____

5. Once installed on a server, how do you ensure confidentiality of the private key? (tick all that apply)

   a. (__) Private key protected by hardware token
   b. (__) Password required for each use of private key
   c. (__) Password required for first use after reboot
   d. (__) Filesystem permissions
   e. (__) No root or administration access over the network. For example if you have configured *sudo(8)* so that no user is unlimited root and only appropriate process has access to the private key.
   f. (__) All system administrators are authorized to access the private key
   g. Other: ______________________________

6. If private key could be stored in a jump start, kick start, or backup image, what confidentiality measures are in place to protect such images? _______________

7. Do you track or register who is authorized to access private keys?

   How: __________________________

   Are there written records? ___________

8. Do you track or register who has system administration access to servers, especially if not all sysadms are authorized to access private keys?

9. Do all those who are authorized to access private keys or who could have access to the private keys (e.g. sysadms) go through training on private keys and sign a confidentiality undertaking regarding them? _________

9.3 Trust Management

1. What is your organization's policy regarding which entities to trust:

   a. (__) Trust anyone
   b. (__) Trust all members of the Trust Network
   c. (__) Trust all members of the Trust Network that also pass local check (e.g. black list)
   d. (__) Explicit local check (e.g. white list)
   e. (__) Other, please describe: ______________

2. What administrative and system administration procedures do you have in place to check that your software is configured to trust only the entities that your organization has decided to trust?

3. What techniques and procedures do you use to ensure that the trust settings are not tampered with and that if tampered, you detect the alterations in a timely manner?

9.4 Threat and Risk Assessments

1. Have you reviewed TAS^3 Threat Analysis document [**?**]?
2. Have you reviewed TAS^3 Risk Assessment document [**?**]?
3. With respect to the services you plan to deploy, which of the mitigation techniques discussed in [**?**] do you plan to implement?

9.5 Service Provider Questions

1. What is your Entity ID? _________________

   Entity ID is decided by you, the organization operating the service. It should be a URL pointing to your SAML metadata. Typically it consists of your domain name, some local path, and possibly of software package dependent part. For example, in

   ```
   https://sp.example.com/svc?o=B
   ```

   the domain name is "sp.example.com", the local path is "/svc" and the product dependent part is "?o=B". The local path depends on how your web server is configured. Consult product documentation for the product dependent part, if any.

2. Does your site support Well Known Location method of SAML metadata exchange (i.e. the metadata is available in the Entity ID URL, consult product documentation if in doubt)?

   (__) Yes, (__) No

   If not, what alternative arrangements do you have for metadata exchange?

3. How do you provide audit drilldown? (check all that apply)

   a. (__) Stand alone web GUI. URL: ________________
   b. (__) iFrame widget Web GUI. URL: ________________
   c. (__) Audit drill down web service (ServiceType "urn:tas3:audit:2010-06")

4. Have you successfully tested sending messages to the Audit Event Bus?

9.5.1 Front End (FE) Single Sign-On Questions

1. Is your software SAML 2.0 compliant? Is it certified? When, by whom: ____
2. Can your software handle ID-WSF 2.0 discovery bootstrap?
3. Which IdPs do you plan to use?
4. Have you exchanged metadata with the IdP?
5. Have you successfully tested SSO with the IdP?

9.5.2 Web Service Provider (WSP) Questions

1. Is your software TAS^3 or ID-WSF 2.0 compliant?

   Is it certified? When, by whom: ____

2. Have you determined

   a. SOAP endpoint URL: ____________________
   b. Human friendly name for your service: ________________
   c. Entity ID of your service (usually different from SOAP endpoint): ____________________________
   d. Service Type URI of your service: ________________________

      The Service Type URI designates the type of service you provide. If you are providing a standardized service, the relevant standard should specify what the Service Type URI is for services of that type. All instances of the service use the same Service Type URI. Some well known Service Types:

      - "urn:ios:pds:2010-05:dst-2.1" - Internet of Subjects Personal Data Store
      - "urn:liberty:id-sis-dap:2006-08:dst-2.1" - Liberty ID Directory Access Protocol
      - "urn:liberty:id-sis-cb:2004-10" - Liberty Contact Book Service
      - "urn:liberty:id-sis-gl:2005-07" - Liberty Geolocation Service
      - "http://www.3gpp.org/ftp/Specs/archive/23_series/23.140/schema/REL-6-MM7-1-4"
        - ID-MM7 messaging service

      If you created the service yourself, you can pick the URI as you please, provided that it is globally unique. The usual convention is to use the namespace URI of the top level XML element of the service payload, i.e. the namespace of the first child element of SOAP Envelope Body element.

3. Have you registered your service end point with a Discovery Service?

   Often the Discovery Service Provider or IdP provides a registration interface on the web. For example the TAS^3 IdP provides "Circle of Trust Manager" at URL `https://idp.tas3.eu/cot/`

   If you do not plan to use discovery, what arrangements do you plan to use to locate your service? What arrangements do you plan to make for issuing security tokens for accessing your service?

4. Have you successfully tested calling your web service from a third party web service client?

5. Is your service an identity service, i.e. does it need to know something about the user?

6. Does your service need persistent handle to user, e.g. to track something about the user (this question aims to establish whether your service needs to see persistent or transient NameID)?

7. What types of credentials need to be presented upon web service call to authorize the call?

   This question aims at determining what credentials your callers will need to gather and present. We do not need full description of your policy.

8. Do you need user to consent to anything and how do you arrange to obtain consent when needed? Do you plan to use the Interaction Service facility and/or handle Interaction Redirect?

9. Are you capable to act as a Credentials and Privacy Negotiation server? If yes, please provide end point URL: _________________

10. What security mechanisms are you willing and able to support

    a. (__) Bearer Token
    b. (__) Holder of Key Token

c. (__) X509 signature without token

d. (__) None

11. Which Policy Enforcement Points do you implement?

a. (__) Request Out PEP

b. (__) Response In PEP

c. (__) Other, please describe: ________________

12. Which Policy Decision Point do you use?

a. (__) Internal or built in

b. (__) External XACML PDP

c. (__) Other: ________________

13. Which obligations or policy languages do you use or support? (tick all that apply)

a. (__) SOL1

b. (__) Permis

c. (__) XACML2

d. (__) Other, please specify: ______________

9.5.3 Attribute Authority Questions

These questions are in addition to the WSP questions of the previous section. You should answer these questions if you are authority for, store, or broker user data, such as Personally Identifiable Information (PII).

1. What is the nature and sensitivity of the user data you handle?

2. What obligations do you pledge to honour with respect to user data trusted in your possession?

   Either describe in prose or provide specific policies using Simple Obligations Language 1 (SOL1) or other obligations language you plan to use.

3. What obligations do you require other party to honour with respect to user data you release?

   Either describe in prose or provide specific policies using Simple Obligations Language 1 (SOL1) or other obligations language you plan to use.

4. Do you have automatic mechanims for satisfying the obligations you pledged? Please describe: ____________________

5. Do you have automatic mechanims for verifying that the requesting party pledges to respect the obligations you issue?

6. What mechanisms do you provide to user and trust network operator to verify that you have complied with your pledges?

7. What mechanisms do you have or require from others to verify that they have complied with their pledges?

8. How do you protect the confidentiality of the stored user data? Describe any filesystem and cryptographic protections you employ.

9. How do you provide Right of Access, Rectification, and Deletion?

   a. (__) Stand alone web GUI. URL: ________________
   b. (__) iFrame widget Web GUI. URL: ________________
   c. (__) Other method: ______________________________

10. In the eventuality of Rectification or Deletion, are you able to notify the parties to whom you have released the data in past?

11. What is your policy towards data requestors who refuse to subscribe to notifications? What about receipients that subscribed, but refuse the actual notification?

9.5.4 Web Service Client (WSC) Questions

A FE or WSP may act in secondary role of Web Service Client (WSC). If you call other web services you should answer these questions.

1. Is your software TAS^3 or ID-WSF 2.0 compliant?

   Is it certified? When, by whom: ____

2. Are you able to use Credentials and Privacy Negotiation agent?

3. Are you able to handle Interaction Redirect if requested by WSP?

4. What security mechanisms are you willing and able to support

   a. (__) Bearer Token
   b. (__) Holder of Key Token
   c. (__) X509 signature without token
   d. (__) None

5. Which Policy Enforcement Points do you implement?

   a. (__) Request Out PEP
   b. (__) Response In PEP
   c. (__) Other, please describe: ________________

6. Which Policy Decision Point do you use?

   a. (__) Internal or built in
   b. (__) External XACML PDP
   c. (__) Other: ________________

7. Which obligations or policy languages do you use or support? (tick all that apply)

   a. (__) SOL1
   b. (__) Permis
   c. (__) XACML2
   d. (__) Other, please specify: ______________

8. What obligations do you pledge to honour with respect to user data returned to you?

   Either describe in prose or provide specific policies using Simple Obligations Language 1 (SOL1) or other obligations language you plan to use.

9. What obligations do you require other party to honour with respect to user data you send?

   Either describe in prose or provide specific policies using Simple Obligations Language 1 (SOL1) or other obligations language you plan to use.

10. Do you have automatic mechanims for satisfying the obligations you pledged? Please describe: ______________________

11. What mechanisms do you provide to user and trust network operator to verify that you have complied with your pledges?

12. What mechanisms do you have or require from others to verify that they have complied with their pledges?

9.6 Single Sign-On Identity Provider (IdP), Discovery Service, Discovery Registry, Identity Mapper, or Delegation Service Questions

1. Is your software SAML 2.0 and TAS^3 or ID-WSF 2.0 compliant?

   Is it certified? When, by whom: ____

2. If your IdP or Discovery Service provides attributes, also answer questions in the Attribute Authority section, above.

9.6.1 Identity Provider Questions

1. What authentication methods do you support (tick all that apply)

   a. (__) One Time Password Token, such as Yubikey, RSA token, or similar
   b. (__) Client certificate at user level or eID card
   c. (__) Mobile phone based authentication
   d. (__) Desktop Login based authentication
   e. (__) Username and password
   f. (__) Other, please specify: ______________________

2. What user intake or vetting procedures do you have?

3. What authentication context classes do you support and how do they map to the intake and authentication methods you support? Please specify the URIs that will be used to indicate these in various protocol transactions.

4. What types of NameIDs are you willing and able to support (tick all that apply)?

   a. (__) Persistent per entity pseudonyms
   b. (__) Transient per entity
   c. (__) Persistent shared unique id (e.g. globally unique id or "national id")
   d. (__) Transient shared (e.g. random ID shared across many entities)

5. Can you push attributes (if you can, you are also an Attribute Authority, see above)?

6. Do you support ID-WSF 2.0 discovery bootstrap attribute?

9.6.2 Discovery Service Questions

1. What registration mechanisms do you provide for WSPs?

   URL of the registration interface: ________________________

2. What security mechanisms are you willing and able to support

   a. (__) Bearer Token
   b. (__) Holder of Key Token
   c. (__) X509 signature without token
   d. (__) None

3. What types of NameIDs are you willing and able to support (tick all that apply)?

   a. (__) Persistent per entity pseudonyms
   b. (__) Transient per entity
   c. (__) Persistent shared unique id (e.g. globally unique id or "national id")
   d. (__) Transient shared (e.g. random ID shared across many entities)

4. Can you push attributes? (if you can you are also an Attribute Authority)

5. Do you support pruning discovery results by trust scoring?

6. Do you support pruning discovery results based on Credentials and Privacy Negotiation?

9.7 Any Other Architectural Role

As other TAS^3 architectural roles are less common and require special considerations, this questionnaire does not try to cover them. Please contact TAS^3 consortium for further guidance.