

PKI applications (C2)

Authentication and Single Sign On

Pascal Steichen (MSSI-uni.lu) - 20/04/2007 (04)

- [1. Authentication](#)
 - [Schneier on Security - The Failure of Two-Factor Authentication](#)
- [1.1. Authentication in PKI](#)
 - [1.2. Beyond authentication](#)
 - [1.3. Kerberos](#)
 - [1.3.1. Kerberos - operation](#)
 - [1.3.2. MS kerberos usage](#)
- [2. Single Sign On](#)
 - [2.1. SSO Implementation examples](#)
 - [2.1.1. Windows Live ID](#)
 - [2.1.2. WS-Trust](#)
 - [2.1.3. Liberty Alliance](#)
 - [2.1.4. SAML](#)
 - [2.1.5. Google](#)
 - [2.1.6. Typo3 SSO](#)
 - [2.1.7. JOSSO](#)
- [3. Identity management](#)
- [4. Bibliographic references](#)

1. Authentication

Authentication

(from Greek authentikos: real or genuine; from authentes: author)

is the act of establishing or confirming something (or someone) as authentic, that is, that claims made by or about the thing are true. Authenticating an object may mean confirming its provenance, whereas authenticating a person often consists of verifying their identity. Authentication depends upon one or more authentication factors.

- The three most commonly recognized factors are:
 - 'something you know' (password, PIN, ...),
 - 'something you have' (credit card, hardware token, ...),
 - 'something you are' (fingerprint, retinal pattern or other biometric).
- Other, less common factors may include:
 - Recognition-based or cognometric authentication
 - where the user has to recognize pre-assigned secret faces
 - Cybermetric authentication
 - Only allowing access from the certain computer, which is the combination of unique hardware and (or) software installed
 - Location-based authentication
 - Only allowing a particular atm, charge, or credit card to be used at a specific merchant or at a specific bank branch, or only allowing root access from specific terminals
 - Time-based authentication
 - Only allowing access from certain accounts during normal working hour
 - Size-based authorization
 - Only allowing a specific transaction to be for a specific exact amount
 - Pre-authorized transactions
 - Where a company uploads all of the check numbers and amounts written for each check to their bank, and the bank would then reject any check not of those numbers and amounts as fraudulent

Using more than one factor is called strong authentication.

Schneier on Security - The Failure of Two-Factor Authentication

March 15, 2005

Two-factor authentication isn't our savior. It won't defend against phishing. It's not going to prevent identity theft. It's not going to secure online accounts from fraudulent transactions. It solves the security problems we had ten years ago, not the security problems we have today.

The problem with passwords is that they're too easy to lose control of. People give them to other people. People write them down, and other people read them. People send them in e-mail, and that e-mail is intercepted. People use them to log into remote servers, and their communications are eavesdropped on. They're also easy to guess. And once any of that happens, the password no longer works as an authentication token because you can't be sure who is typing that password

in.

Two-factor authentication mitigates this problem. If your password includes a number that changes every minute, or a unique reply to a random challenge, then it's harder for someone else to intercept. You can't write down the ever-changing part. An intercepted password won't be good the next time it's needed. And a two-factor password is harder to guess. Sure, someone can always give his password and token to his secretary, but no solution is foolproof.

These tokens have been around for at least two decades, but it's only recently that they have gotten mass-market attention. AOL is rolling them out. Some banks are issuing them to customers, and even more are talking about doing it. It seems that corporations are finally waking up to the fact that passwords don't provide adequate security, and are hoping that two-factor authentication will fix their problems.

Unfortunately, the nature of attacks has changed over those two decades. Back then, the threats were all passive: eavesdropping and offline password guessing. Today, the threats are more active: phishing and Trojan horses.

Here are two new active attacks we're starting to see:

* Man-in-the-Middle attack. An attacker puts up a fake bank website and entices user to that website. User types in his password, and the attacker in turn uses it to access the bank's real website. Done right, the user will never realize that he isn't at the bank's website. Then the attacker either disconnects the user and makes any fraudulent transactions he wants, or passes along the user's banking transactions while making his own transactions at the same time.

* Trojan attack. Attacker gets Trojan installed on user's computer. When user logs into his bank's website, the attacker piggybacks on that session via the Trojan to make any fraudulent transaction he wants.

See how two-factor authentication doesn't solve anything? In the first case, the attacker can pass the ever-changing part of the password to the bank along with the never-changing part. And in the second case, the attacker is relying on the user to log in.

The real threat is fraud due to impersonation, and the tactics of impersonation will change in response to the defenses. Two-factor authentication will force criminals to modify their tactics, that's all.

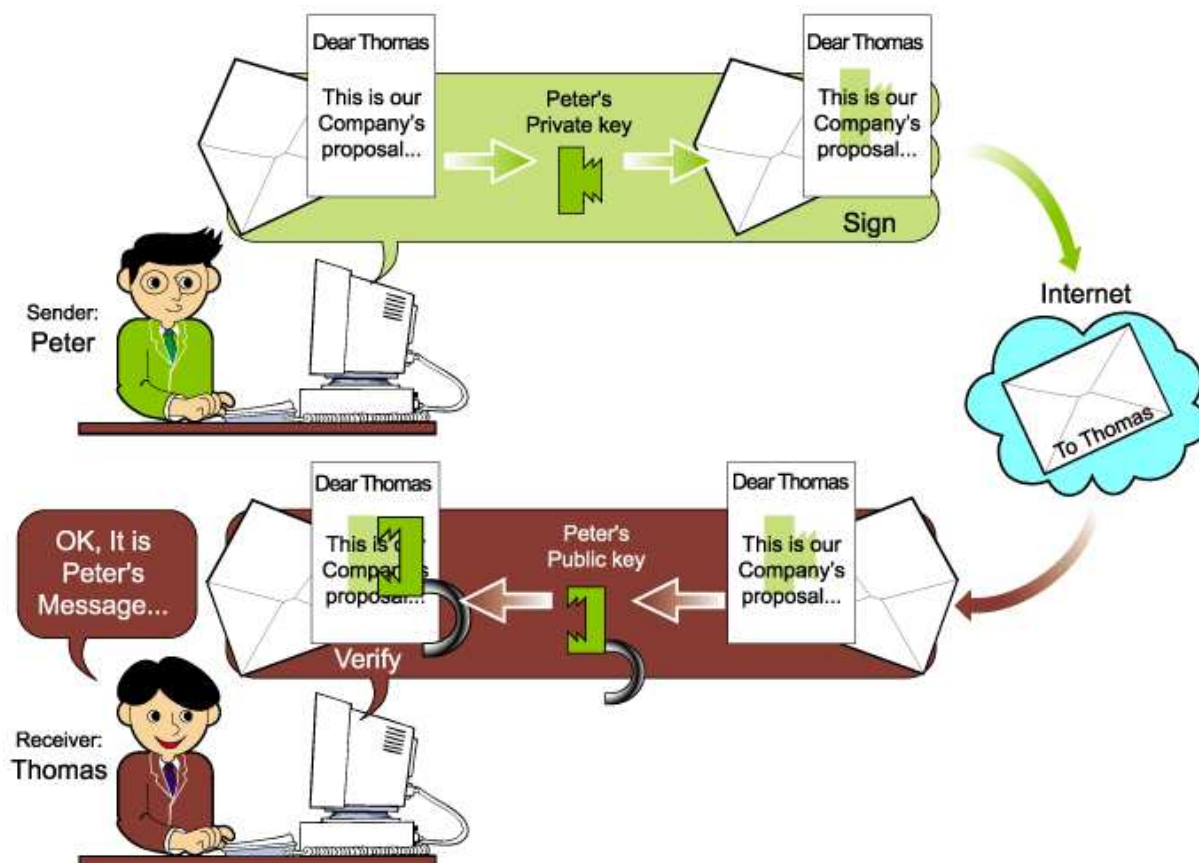
Recently I've seen examples of two-factor authentication using two different communications paths: call it "two-channel authentication." One bank sends a challenge to the user's cell phone via SMS and expects a reply via SMS. If you assume that all your customers have cell phones, then this results in a two-factor authentication process without extra hardware. And even better, the second authentication piece goes over a different communications channel than the first; eavesdropping is much, much harder.

But in this new world of active attacks, no one cares. An attacker using a man-in-the-middle attack is happy to have the user deal with the SMS portion of the log-in, since he can't do it himself. And a Trojan attacker doesn't care, because he's relying on the user to log in anyway.

Two-factor authentication is not useless. It works for local login, and it works within some corporate networks. But it won't work for remote authentication over the Internet. I predict that banks and other financial institutions will spend millions outfitting their users with two-factor authentication tokens. Early adopters of this technology may very well experience a significant drop in fraud for a while as attackers move to easier targets, but in the end there will be a negligible drop in the amount of fraud and identity theft.

© 2005 Bruce Schneier

1.1. Authentication in PKI



Department of Computer Science and Information Systems, The University of Hong Kong

1.2. Beyond authentication

Triple A:

- Authentication

Authentication refers to the confirmation that a user who is requesting services is a valid user of the network services requested. Authentication is accomplished via the presentation of an identity and credentials. Examples of types of credentials are passwords, one-time tokens, digital certificates, and phone numbers (calling/called).

- Authorization

Authorization refers to the granting of specific types of service (including "no service") to a user, based on their authentication, what services they are requesting, and the current system state. Authorization may be based on restrictions, for example time-of-day restrictions, or physical location restrictions, or restrictions against multiple logins by the same user. Authorization determines the nature of the service which is granted to a user. Examples of types of service include, but are not limited to: IP address filtering, address assignment, route assignment, QoS/differential services, bandwidth control/traffic management, compulsory tunneling to a specific endpoint, and encryption.

- Accounting

Accounting refers to the tracking of the consumption of network resources by users. This information may be used for management, planning, billing, or other purposes. Real-time accounting refers to accounting information that is delivered concurrently with the consumption of the resources. Batch accounting refers to accounting information that is saved until it is delivered at a later time. Typical information that is gathered in accounting is the identity of the user, the nature of the service delivered, when the service began, and when it ended.

- Audit

The AAA is sometimes combined with auditing and accordingly becomes AAAA

1.3. Kerberos



MIT developed Kerberos to protect network services provided by Project Athena. The protocol was named after the Greek mythological character Kerberos (or Cerberus), known in Greek mythology as being the monstrous three-headed guard dog of Hades.

- Kerberos is a computer network authentication protocol

which allows individuals communicating over an insecure network to prove their identity to one another in a secure manner. Kerberos prevents eavesdropping or replay attacks, and ensures the integrity of the data. Its designers aimed primarily at a client-server model, and it provides mutual authentication - both the user and the server verify each other's identity.

- Version 5 is now (2005) available as RFC 4120.
- Kerberos builds on symmetric key cryptography and requires a trusted third party.

Kerberos uses as its basis the Needham-Schroeder protocol. It makes use of a trusted third party, termed:

- Key Distribution Center (KDC)

which consists of two logically separate parts:

- Authentication Server (AS)
- Ticket Granting Server (TGS)

providing 'access' to

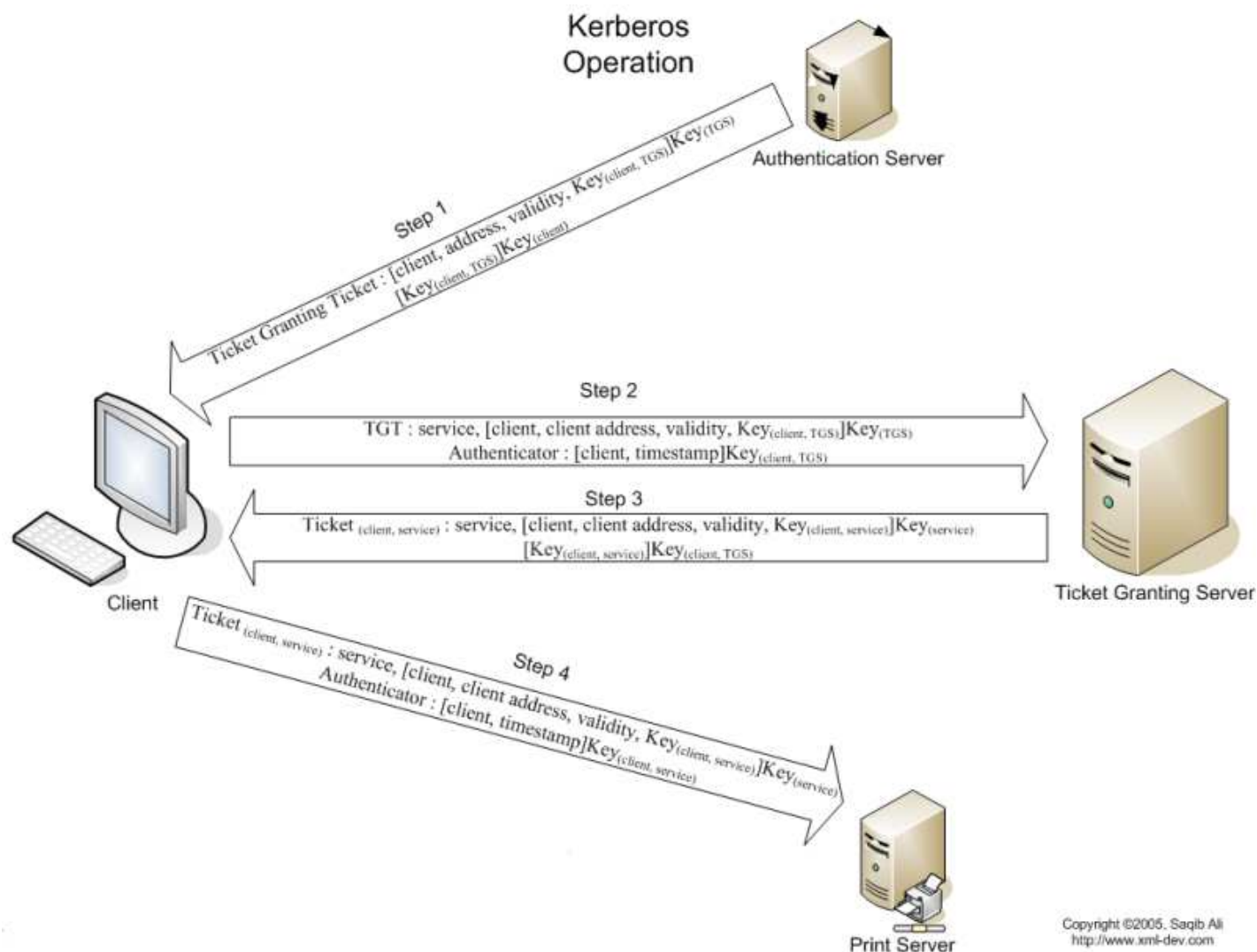
- Service Server (SS)

Kerberos works on the basis of "tickets" which serve to prove the identity of users.

The KDC maintains a database of secret keys; each entity on the network - whether a client or a server - shares a secret key known only to itself and to the KDC. Knowledge of this key serves to prove an entity's identity. For communication between two entities, the KDC generates a session key which they can use to secure their interactions.

1.3.1. Kerberos - operation

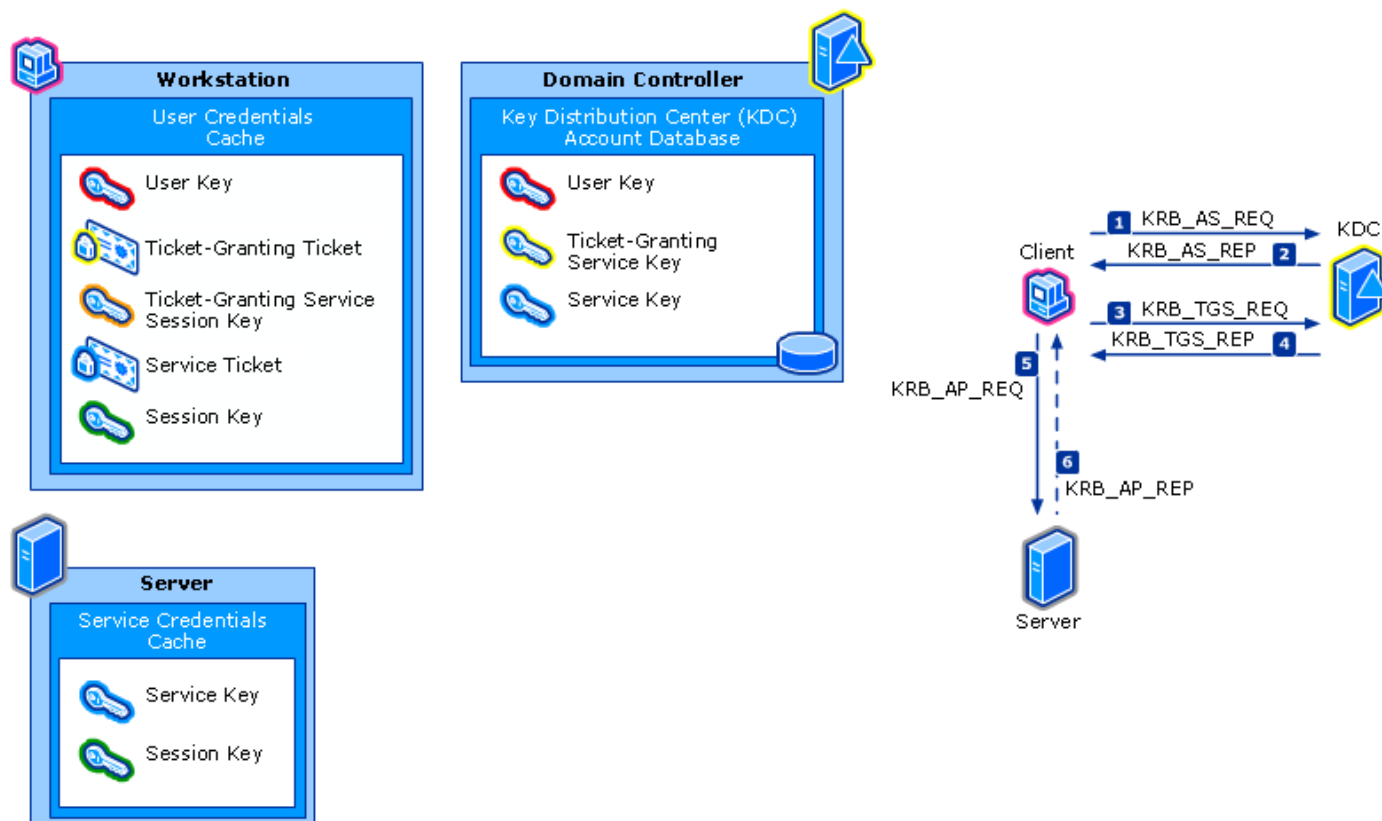
In one sentence: the client authenticates itself to the "authentication server" (AS), then demonstrates to the "ticket granting server" (TGS) that it's authorized to receive a ticket for a service (and receives it), then demonstrates to the "service server" (SS) that it has been approved to receive the service.



In more detail:

1. A user enters a username and password on the client.
2. The client performs a one-way hash on the entered password, and this becomes the secret key of the client.
3. The client sends a clear-text message to the AS requesting services on behalf of the user. Sample Message: "User XYZ would like to request services". Note: Neither the secret key nor the password is sent to the AS.
4. The AS checks to see if the client is in its database. If it is, the AS sends back the following two messages to the client:
 - Message A: Client/TGS session key encrypted using the secret key of the user.
 - Message B: Ticket-Granting Ticket (which includes the client ID, client network address, ticket validity period, and the client/TGS session key) encrypted using the secret key of the TGS.
5. Once the client receives messages A and B, it decrypts message A to obtain the client/TGS session key. This session key is used for further communications with TGS. (Note: The client cannot decrypt the Message B, as it is encrypted using TGS's secret key.) At this point, the client has enough information to authenticate itself to the TGS.
6. When requesting services, the client sends the following two messages to the TGS:
 - Message C: Composed of the Ticket-Granting Ticket from message B and the ID of the requested service.
 - Message D: Authenticator (which is composed of the client ID and the timestamp), encrypted using the client/TGS session key.
7. Upon receiving messages C and D, the TGS decrypts message D (Authenticator) using the client/TGS session key and sends the following two messages to the client:
 - Message E: Client-to-server ticket (which includes the client ID, client network address, validity period and Client/server session key) encrypted using the service's secret key.
 - Message F: Client/server session key encrypted with the client/TGS session key.
8. Upon receiving messages E and F from TGS, the client has enough information to authenticate itself to the SS. The client connects to the SS and sends the following two messages:
 - Message E from the previous step (the client-to-server ticket, encrypted using service's secret key).
 - Message G: a new Authenticator, which includes the client ID, timestamp and is encrypted using client/server session key.
9. The SS decrypts the ticket using its own secret key and sends the following message to the client to confirm its true identity and willingness to serve the client:
 - Message H: the timestamp found in client's recent Authenticator plus 1, encrypted using the client/server session key.
10. The client decrypts the confirmation using its shared key with the server and checks whether the timestamp is correctly updated. If so, then the client can trust the server and can start issuing service requests to the server.
11. The server provides the requested services to the client.

1.3.2. MS kerberos usage



© 2003 Microsoft

- The Authentication Service Exchange

1. Kerberos authentication service request (KRB_AS_REQ)

The client contacts the Key Distribution Center's authentication service for a short-lived ticket (a message containing the client's identity and - for Windows client's - SIDs) called a ticket-granting ticket (TGT). This happens at logon.

2. Kerberos authentication service response (KRB_AS_REP)

The authentication service (AS) constructs the TGT and creates a session key the client can use to encrypt communication with the ticket-granting service (TGS). The TGT has a limited lifetime. At the point that the client has received the TGT, the client has not been granted access to any resources, even to resources on the local computer.

Why use a TGT? Couldn't the AS simply issue a ticket for the target server? Yes, but if the AS issued tickets directly, the user would have to enter a password for every new server/service connection. Issuing a TGT with a short lifespan (typically 10 hours) gives the user a valid ticket for the ticket-granting service, which in turn issues target-server tickets. The TGT's main benefit is that the user only has to enter a password once, at logon.

- The Ticket-Granting Service Exchange

3. Kerberos ticket-granting service request (KRB_TGS_REQ)

The client wants access to local and network resources. To gain access, the client sends a request to the TGS for a ticket for the local computer or some network server or service. This ticket is referred to as the service ticket or service ticket. To get the ticket, the client presents the TGT, an authenticator, and the name of the target server (the Server Principal Name or SPN).

4. Kerberos ticket-granting service response (KRB_TGS_REP)

The TGS examines the TGT and the authenticator. If these are acceptable, the TGS creates a service ticket. The client's identity is taken from the TGT and copied to the service ticket. Then the ticket is sent to the client.

The TGS cannot determine if the user will be able to get access to the target server. It simply returns a valid ticket. Authentication does not imply authorization.

- The Client/Server Exchange

5. Kerberos application server request (KRB_AP_REQ)

After the client has the service ticket, the client sends the ticket and a new authenticator to the target server, requesting access. The server will decrypt the ticket, validate the authenticator, and for Windows services, create an access token for the user based on the SIDs in the ticket.

6. Kerberos application server response (optional) (KRB_AP_REP)

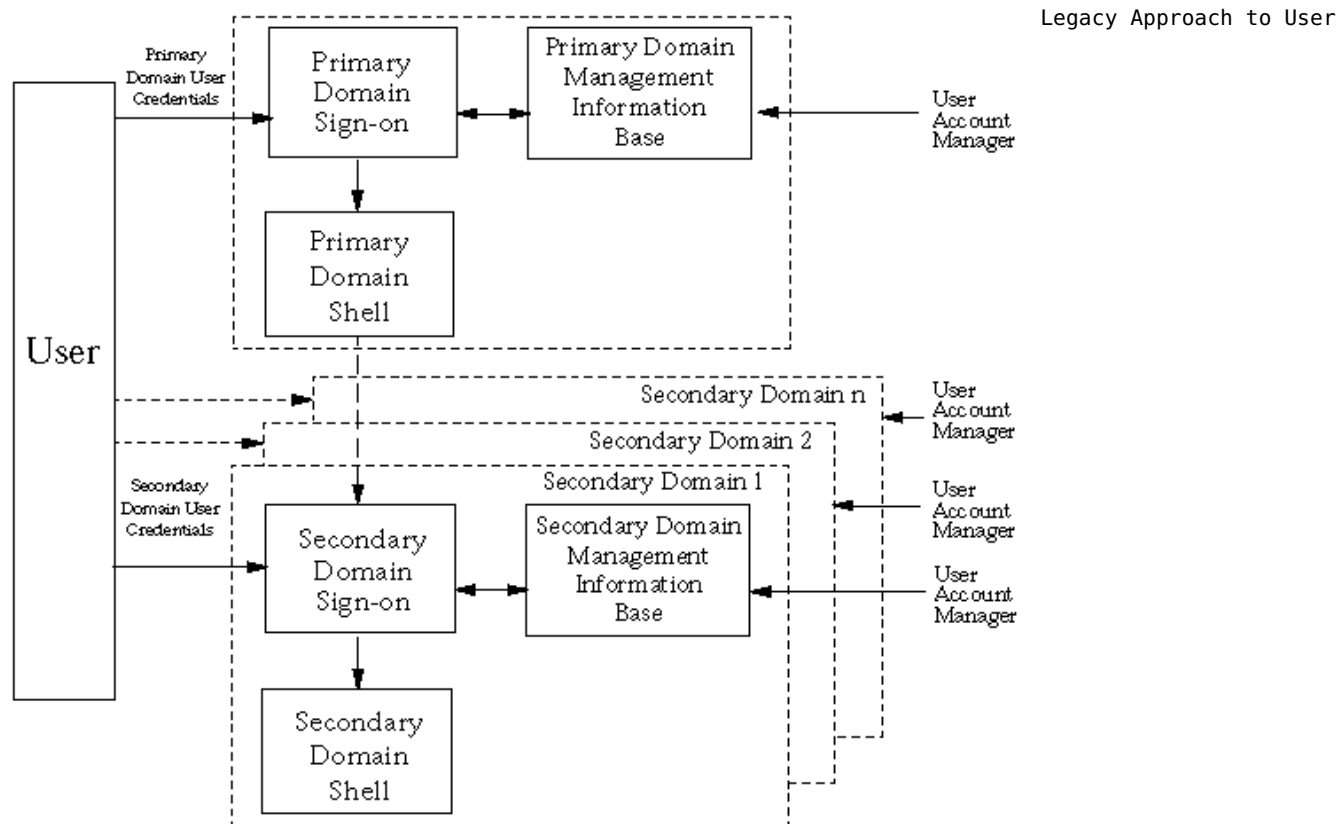
Optionally, the client might request that the target server verify its own identity. This is called mutual authentication. If mutual authentication is requested, the target server will take the client computer's timestamp from the authenticator,

encrypt it with the session key the TGS provided for client-target server messages, and send it to the client.

2. Single Sign On

A specialized form of authentication that enables a user to authenticate **once** and gain access to the **multiple** resources.

As IT systems proliferate to support business processes, users and system administrators are faced with an increasingly complicated interface to accomplish their job functions. Users typically have to sign-on to multiple systems, necessitating an equivalent number of sign-on dialogues, each of which may involve different usernames and authentication information. System administrators are faced with managing user accounts within each of the multiple systems to be accessed in a co-ordinated manner in order to maintain the integrity of security policy enforcement. This legacy approach to user sign-on to multiple systems is illustrated below:

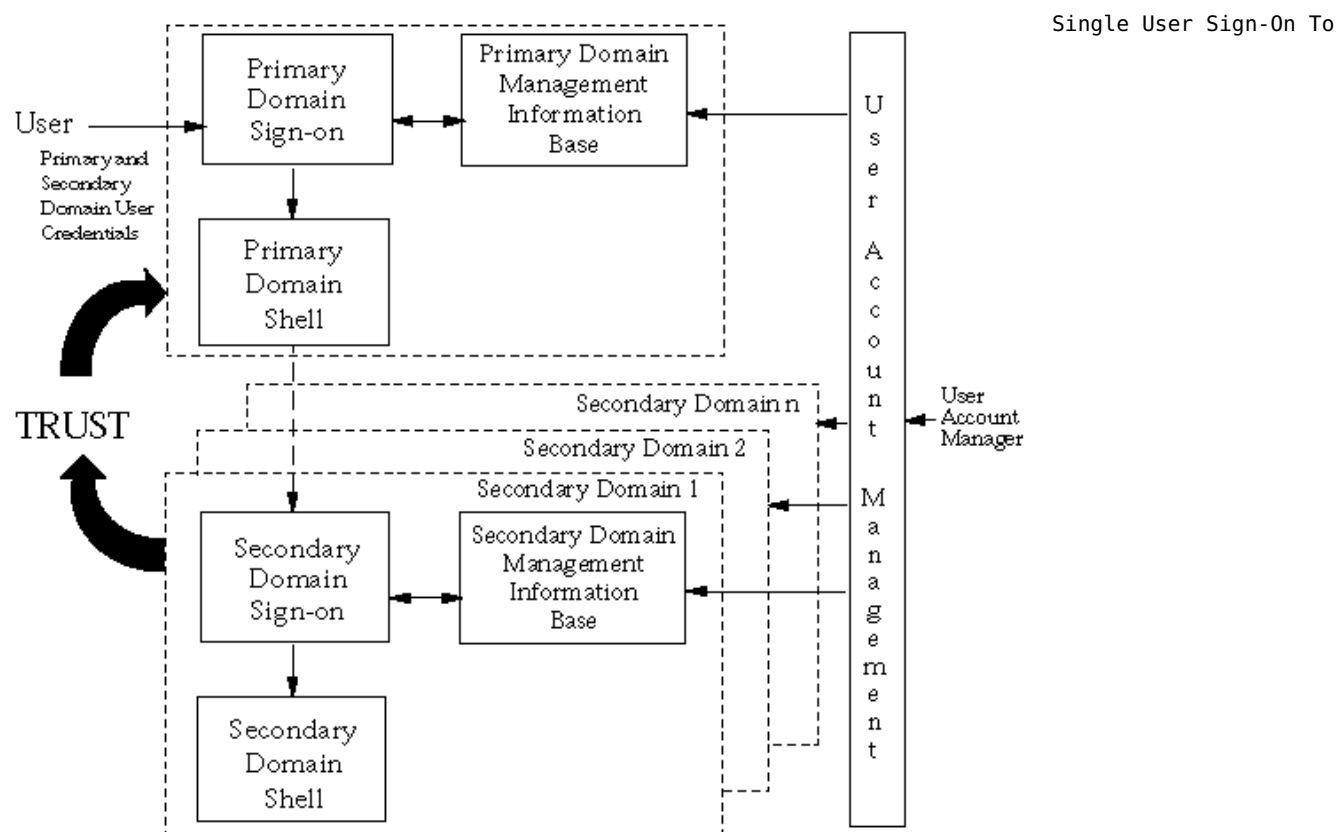


Sign-on to Multiple Systems - © 1995-2005 The OpenGroup

Historically a distributed system has been assembled from components that act as independent security domains. These components comprise individual platforms with associated operating system and applications.

These components act as independent domains in the sense that an end-user has to identify and authenticate himself independently to each of the domains with which he wishes to interact. This scenario is illustrated above. The end user interacts initially with a Primary Domain to establish a session with that primary domain. This is termed the Primary Domain Sign-On in the above diagram and requires the end user to supply a set of user credentials applicable to the primary domain, for example a username and password. The primary domain session is typically represented by an operating system session shell executed on the end user's workstation within an environment representative of the end user (e.g., process attributes, environment variables and home directory). From this primary domain session shell the user is able to invoke the services of the other domains, such as platforms or applications.

To invoke the services of a secondary domain an end user is required to perform a Secondary Domain Sign-on. This requires the end user to supply a further set of user credentials applicable to that secondary domain. An end user has to conduct a separate sign-on dialogue with each secondary domain that the end user requires to use. The secondary domain session is typically represented by an operating system shell or an application shell, again within an environment representative of the end user. From the management perspective the legacy approach requires independent management of each domain and the use of multiple user account management interfaces. Considerations of both usability and security give rise to a need to co-ordinate and where possible integrate user sign-on functions and user account management functions for the multitude of different domains now found within an enterprise. A service that provides such co-ordination and integration can provide real cost benefits to an enterprise through:



Multiple Services - © 1995-2005 The OpenGroup

- reduction in the time taken by users in sign-on operations to individual domains, including reducing the possibility of such sign-on operations failing
- improved security through the reduced need for a user to handle and remember multiple sets of authentication information,
- reduction in the time taken, and improved response, by system administrators in adding and removing users to the system or modifying their access rights,
- improved security through the enhanced ability of system administrators to maintain the integrity of user account configuration including the ability to inhibit or remove an individual user's access to all system resources in a co-ordinated and consistent manner.

Such a service has been termed Single Sign-On after the end-user perception of the impact of this service. However, both the end-user and management aspects of the service are equally important. This approach is illustrated in the diagram above. In the single sign-on approach the system is required to collect from the user as part of the primary sign-on, all the identification and user credential information necessary to support the authentication of the user to each of the secondary domains that the user may potentially require to interact with. The information supplied by the user is then used by Single Sign-On Services within the primary domain to support the authentication of the end user to each of the secondary domains with which the user actually requests to interact.

The information supplied by the end-user as part of the Primary Domain Sign-On procedure may be used in support of secondary domain sign-on in several ways:

- **directly**, the information supplied by the user is passed to a secondary domain as part of a secondary sign-on,
- **indirectly**, the information supplied by the user is used to retrieve other user identification and user credential information stored within the a single sign-on management information base. The retrieved information is then used as the basis for a secondary domain sign-on operation,
- **immediately**, to establish a session with a secondary domain as part of the initial session establishment. This implies that application clients are automatically invoked and communications established at the time of the primary sign-on operation,
- **temporarily** stored or cached and used at the time a request for the secondary domain services is made by the end-user.

From a management perspective the single sign-on model provides a single user account management interface through which all the component domains may be managed in a coordinated and synchronised manner.

Significant security aspects of the Single Sign-On model are:

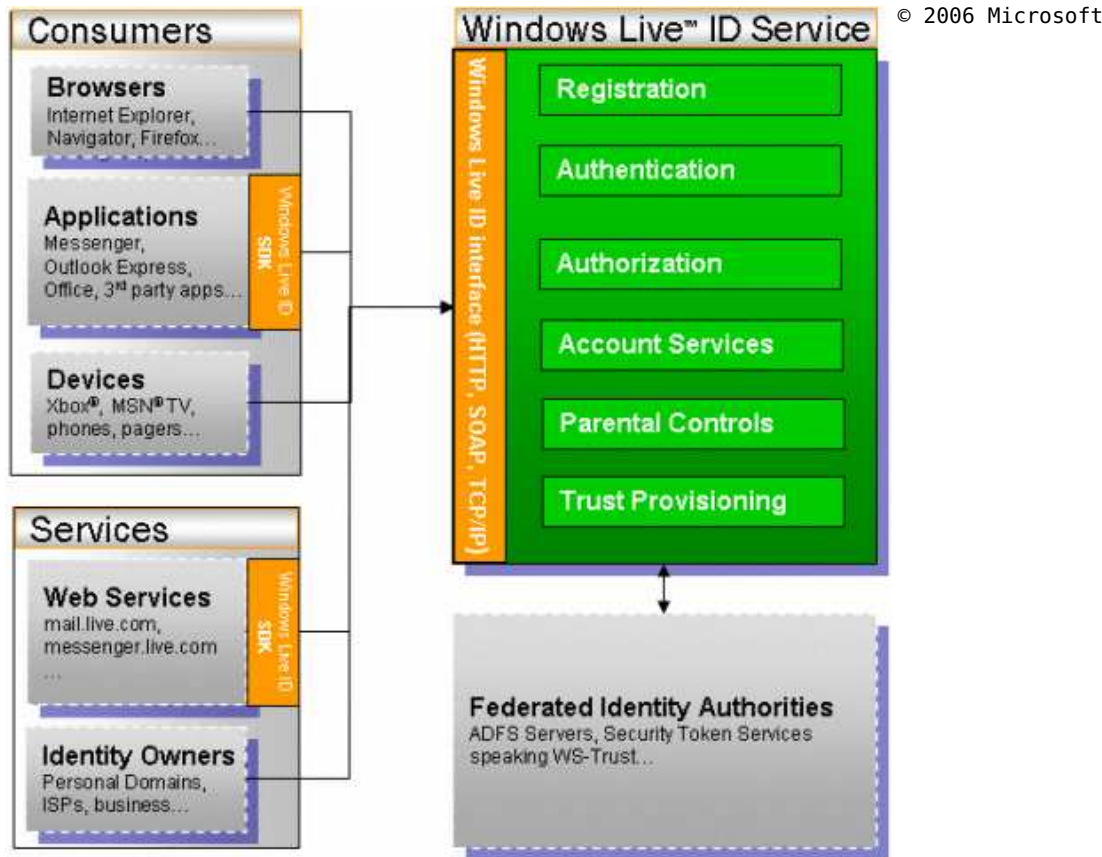
- the secondary domains have to trust the primary domain to:
 - correctly assert the identity and authentication credentials of the end user,
 - protect the authentication credentials used to verify the end user identity to the secondary domain from unauthorised use,
- the authentication credentials have to be protected when transferred between the primary and secondary domains against threats arising from interception or eavesdropping leading to possible masquerade attacks.

2.1. SSO implementation examples

- Windows Live ID (ex .NET Passport)
- Liberty Alliance
- SAML
- Google
- Typo3 SSO
- JOSSO

2.1.1. Windows Live ID

Windows Live ID (originally named .NET Passport; briefly Microsoft Passport Network) is a "unified-login" service developed and provided by Microsoft that allows users to log in to many websites using one account. It was originally positioned as a single sign-on service for all web commerce.



2.1.2. WS-Trust

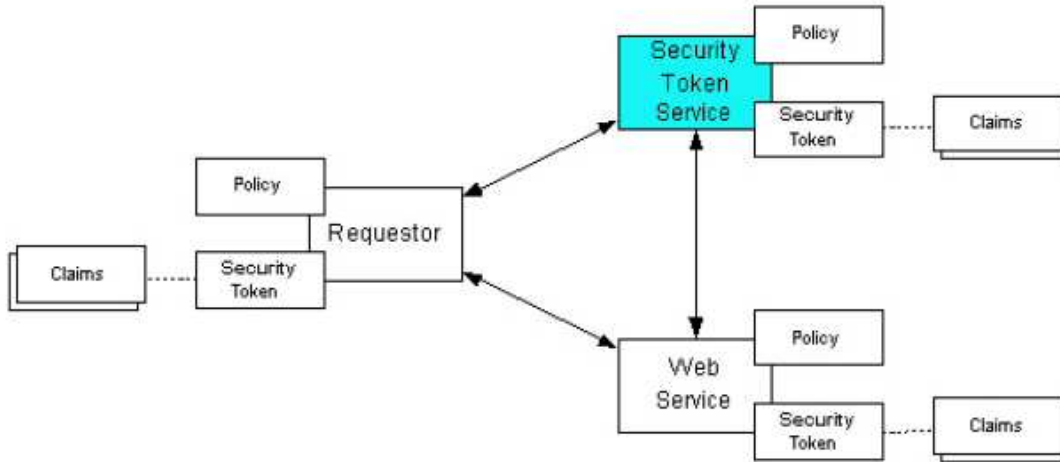
The Web service security model defined in WS-Trust is based on a process in which a Web service can require that an incoming message prove a set of claims (e.g., name, key, permission, capability, etc.). If a message arrives without having the required proof of claims, the service SHOULD ignore or reject the message. A service can indicate its required claims and related information in its policy as described by [WS-Policy] and [WS-PolicyAttachment] specifications.

Authentication of requests is based on a combination of optional network and transport-provided security and information (claims) proven in the message. Requestors can authenticate recipients using network and transport-provided security, claims proven in messages, and encryption of the request using a key known to the recipient.

One way to demonstrate authorized use of a security token is to include a digital signature using the associated secret key (from a proof-of-possession token). This allows a requestor to prove a required set of claims by associating security tokens (e.g., PKIX, X.509 certificates) with the messages.

- If the requestor does not have the necessary token(s) to prove required claims to a service, it can contact appropriate authorities (as indicated in the service's policy) and request the needed tokens with the proper claims. These "authorities", which we refer to as security token services, may in turn require their own set of claims for authenticating and authorizing the request for security tokens. Security token services form the basis of trust by issuing a range of security tokens that can be used to broker trust relationships between different trust domains.
- This specification also defines a general mechanism for multi-message exchanges during token acquisition. One example use of this is a challenge-response protocol that is also defined in this specification. This is used by a Web service for additional challenges to a requestor to ensure message freshness and verification of authorized use of a security token.

This model is illustrated in the figure below, showing that any requestor may also be a service, and that the Security Token Service is a Web service (that is, it may express policy and require security tokens).

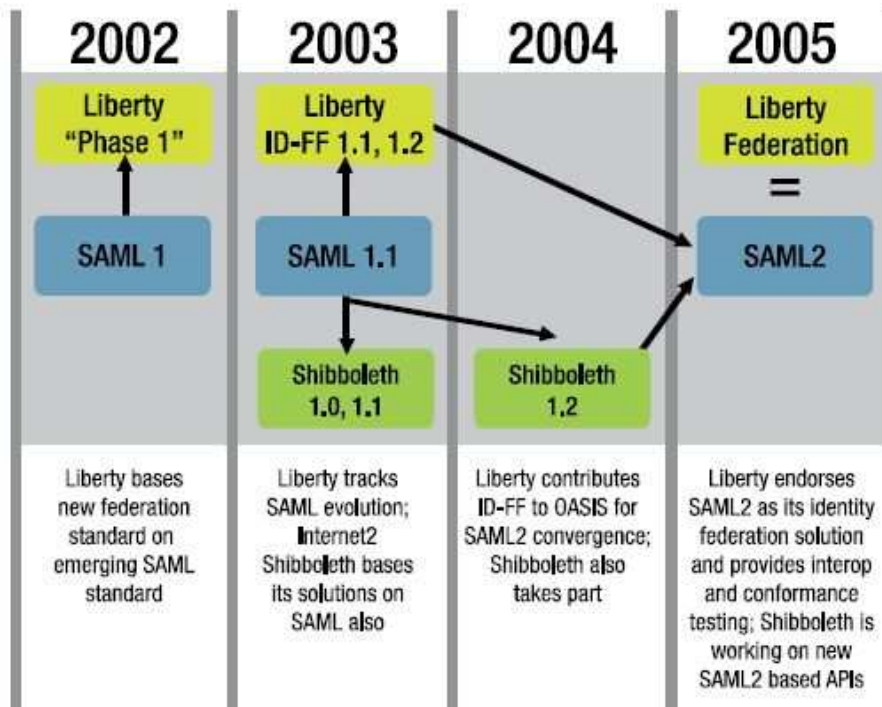


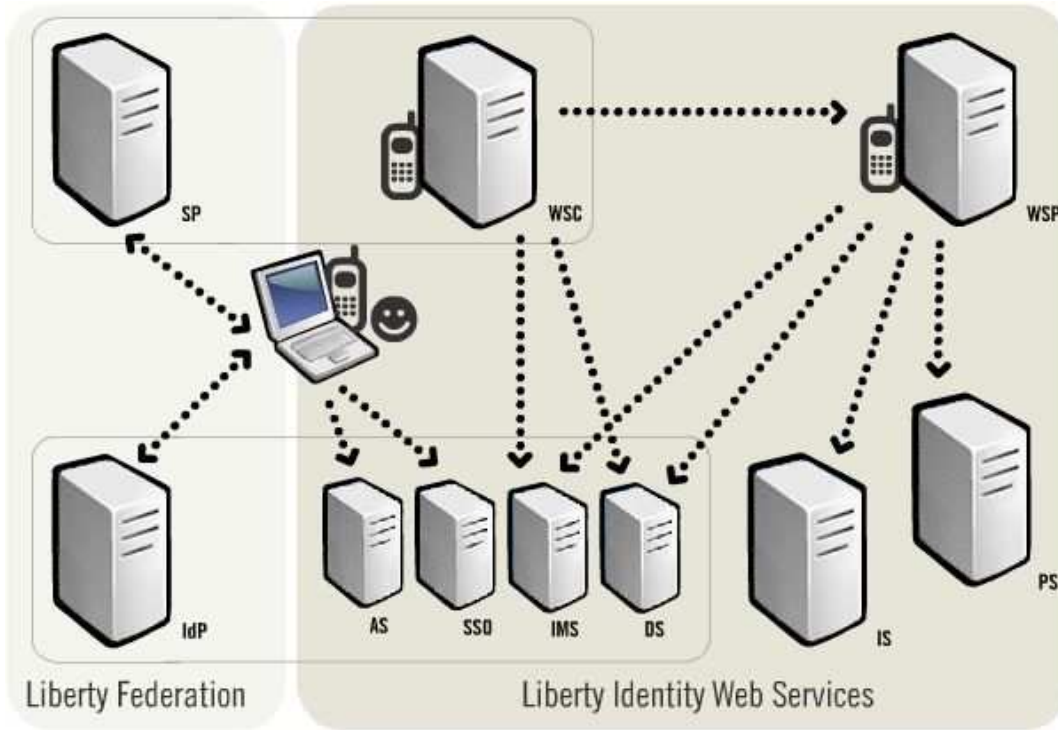
2.1.3. Liberty Alliance

The vision of Liberty Alliance is to enable a networked world based on open standards where consumers, citizens, businesses and governments can more easily conduct online transactions while protecting the privacy and security of identity information. This world, where devices and identities of all kinds are linked by federation and protected by universal strong authentication, is being built today with Liberty's open identity standards, business and deployment guidelines and best practices for managing privacy.

Members work closely together to:

- Build open standard-based specifications for federated identity and identity-based Web services.
- Drive global identity theft solutions.
- Provide interoperability testing.
- Offer a formal certification program for products utilizing Liberty specifications.
- Establish best practices, rules, liabilities, and business guidelines.
- Collaborate with other standards bodies, privacy advocates, and government groups.
- Address end user privacy and confidentiality issues.



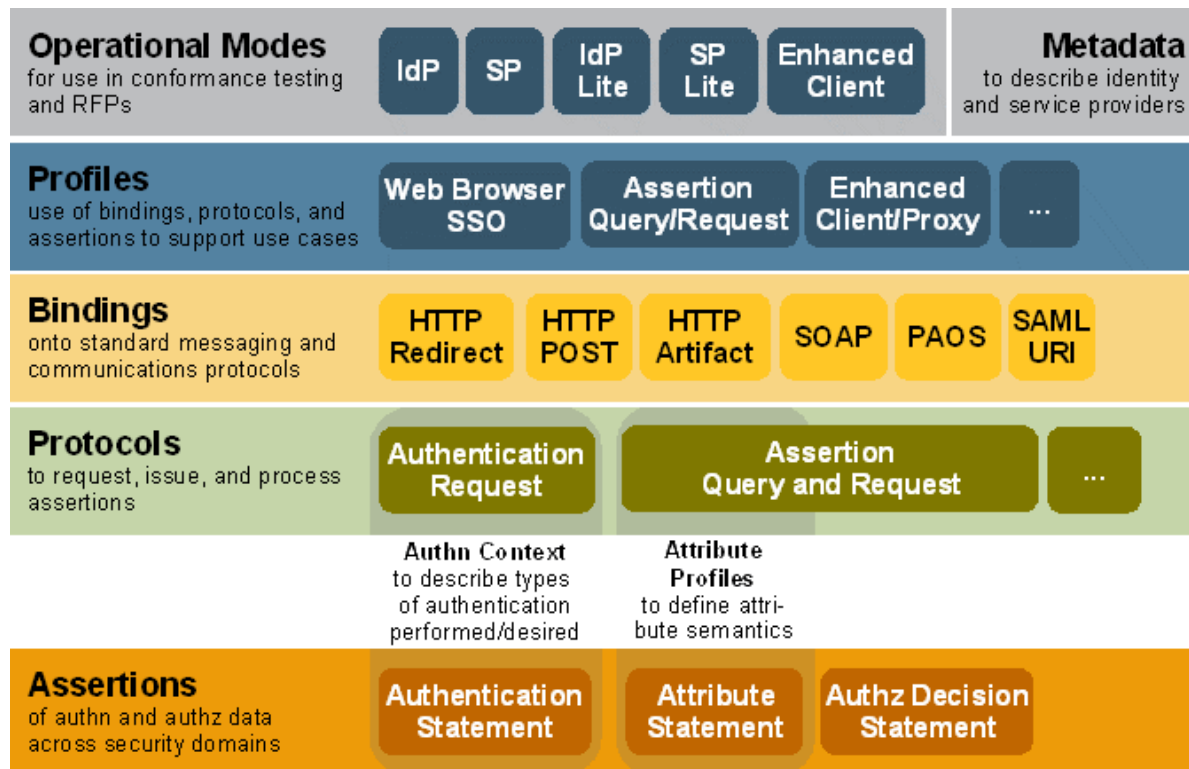


2.1.4. SAML

Security Assertion Markup Language (SAML) is an XML standard for exchanging authentication and authorization data between security domains, that is, between an identity provider (a producer of assertions) and a service provider (a consumer of assertions). SAML is a product of the OASIS Security Services Technical Committee.

SAML assumes the principal (often a user) has enrolled with at least one identity provider. This identity provider is expected to provide local authentication services to the principal. However, SAML does not specify the implementation of these local services; indeed, SAML does not care how local authentication services are implemented (although individual service providers most certainly will).

Thus a service provider relies on the identity provider to identify the principal. At the principal's request, the identity provider passes a SAML assertion to the service provider. On the basis of this assertion, the service provider makes an access control decision.



SAML assertions are usually transferred from identity providers to service providers. Assertions contain statements that service

providers use to make access control decisions. Three types of statements are provided by SAML:

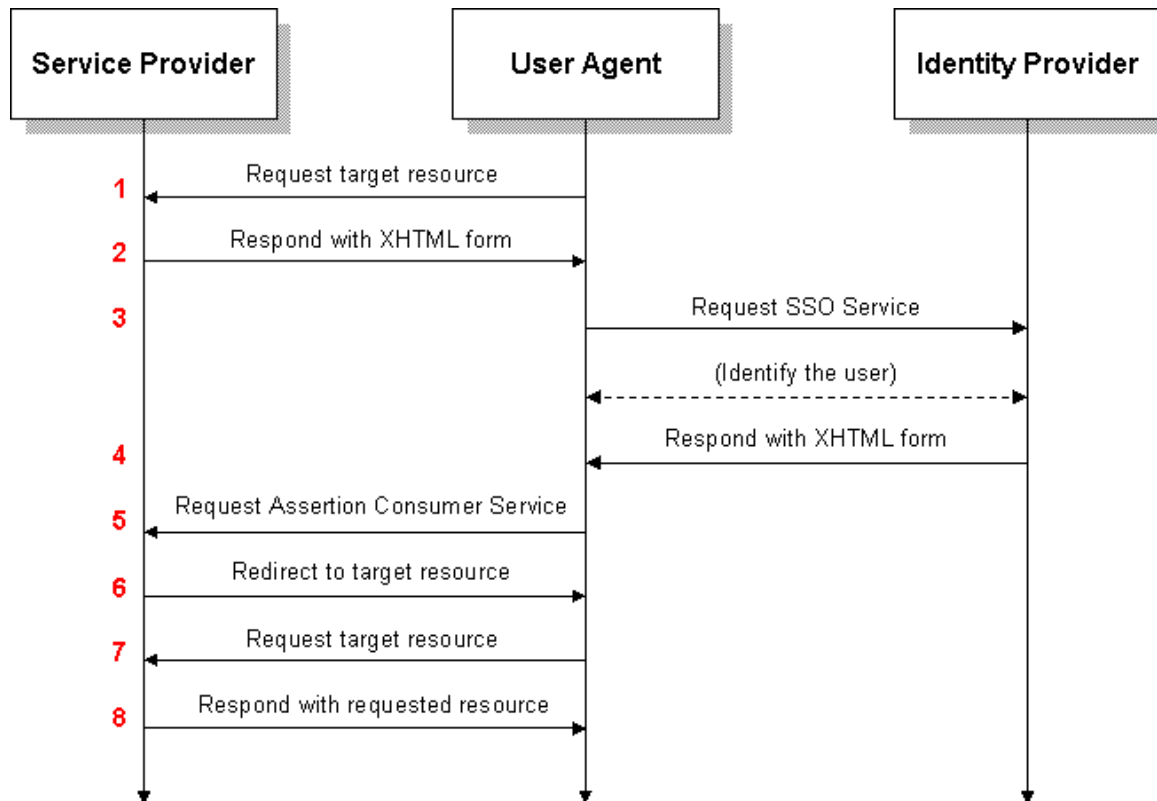
1. Authentication statements
2. Attribute statements
3. Authorization decision statements

Authentication statements assert to the service provider that the principal did indeed authenticate with the identity provider at a particular time using a particular method of authentication. Other information about the authenticated principal (called the authentication context) may be disclosed in an authentication statement.

An attribute statement asserts that a subject is associated with certain attributes. An attribute is simply a name-value pair. Relying parties use attributes to make access control decisions.

An authorization decision statement asserts that a subject is permitted to perform action A on resource R given evidence E. The expressiveness of authorization decision statements in SAML is intentionally limited.

The primary SAML use case is called Web Browser Single Sign-On (SSO). A user wielding a user agent (usually a web browser) requests a web resource protected by a SAML service provider. The service provider, wishing to know the identity of the requesting user, issues an authentication request to a SAML identity provider through the user agent. The resulting protocol flow is depicted in the following diagram.



1. Request the target resource at the SP

The principal (via an HTTP user agent) requests a target resource at the service provider:

```
https://sp.example.com/myresource
```

The service provider performs a security check on behalf of the target resource. If a valid security context at the service provider already exists, skip steps 2-7.

2. Respond with an XHTML form

The service provider responds with a document containing an XHTML form:

```
<form method="post" action="https://idp.example.org/SAML2/SSO/POST" ...>
  <input type="hidden" name="SAMLRequest" value="request" />
  ...
  <input type="submit" value="Submit" />
</form>
```

The value of the SAMLRequest parameter is the base64 encoding of a `<samlp:AuthnRequest>` element.

3. Request the SSO Service at the IdP

The user agent issues a POST request to the SSO service at the identity provider where the value of the SAMLRequest parameter is taken from the XHTML form at step 2. The SSO service processes the AuthnRequest and performs a security check. If the user does not have a valid security context, the identity provider identifies the user (details omitted).

4. Respond with an XHTML form

The SSO service validates the request and responds with a document containing an XHTML form:

```
<form method="post" action="https://sp.example.com/SAML2/SSO/POST" ...>
  <input type="hidden" name="SAMLResponse" value="response" />
  ...
  <input type="submit" value="Submit" />
</form>
```

The value of the SAMLResponse parameter is the base64 encoding of a <samlp:Response> element.

5. Request the Assertion Consumer Service at the SP

The user agent issues a POST request to the assertion consumer service at the service provider. The value of the SAMLResponse parameter is taken from the XHTML form at step 4.

6. Redirect to the target resource

The assertion consumer service processes the response, creates a security context at the service provider and redirects the user agent to the target resource.

7. Request the target resource at the SP again

The user agent requests the target resource at the service provider (again):

```
https://sp.example.com/myresource
```

8. Respond with requested resource

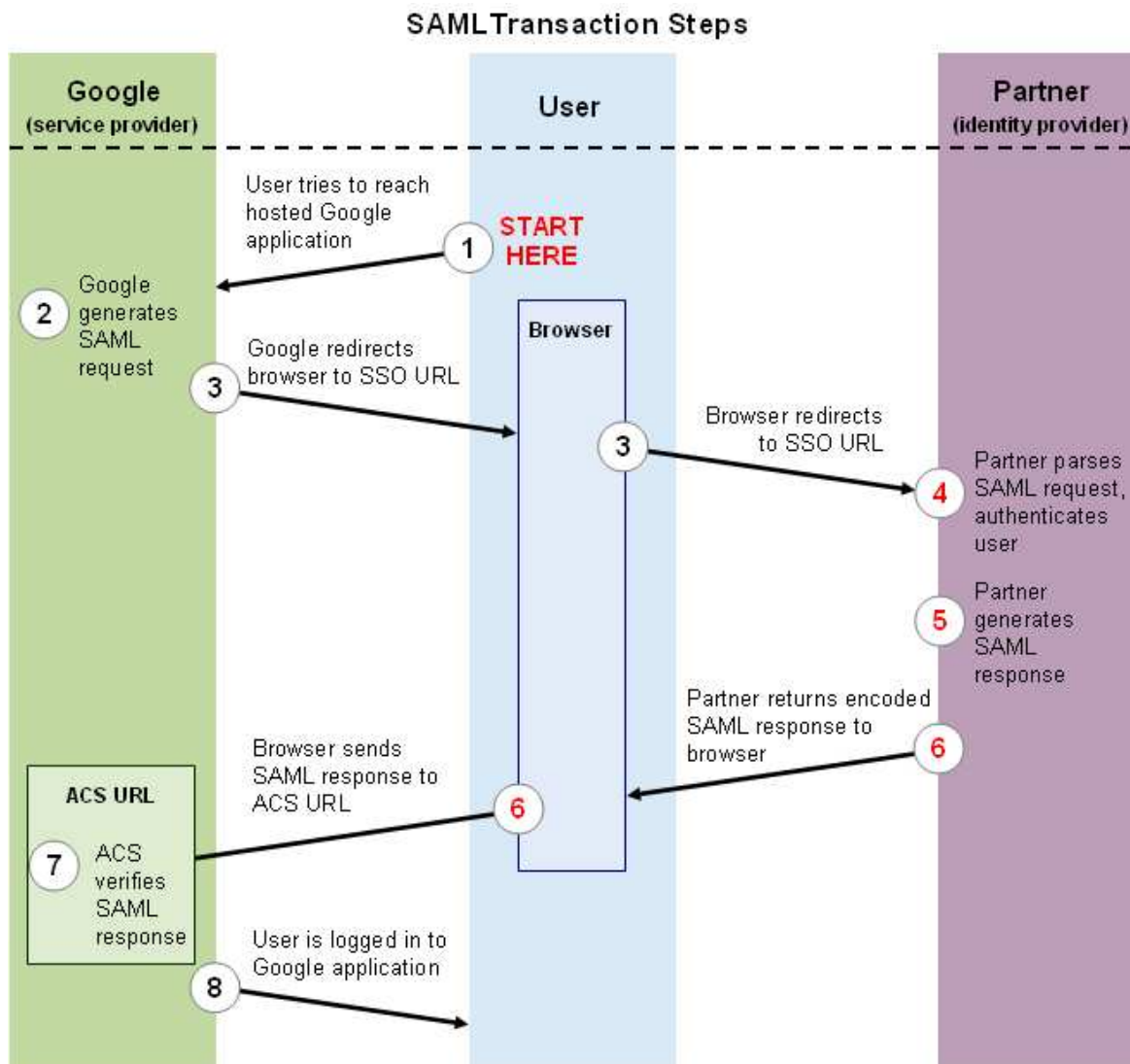
Since a security context exists, the service provider returns the resource to the user agent.

2.1.5. Google

Google Apps offers a SAML-based Single Sign-On (SSO) service that provides partner companies with full control over the authorization and authentication of hosted user accounts that can access web-based applications like Gmail or Google Calendar. Using the SAML model, Google acts as the service provider and provides services such as Gmail and Partner Start Pages (PSP). Google partners act as identity providers and control usernames, passwords and other information used to identify, authenticate and authorize users for web applications that Google hosts.

It is important to note that the SSO solution only applies to web applications. If you want to enable your users to access Google services with desktop clients such as Outlook - for example, Outlook would provide POP access to Gmail - you will still need to provide your users with usable passwords and synchronize those passwords with your internal user database using the Provisioning API.

The following process explains how a user logs into a hosted Google application through a partner-operated, SAML-based SSO service.

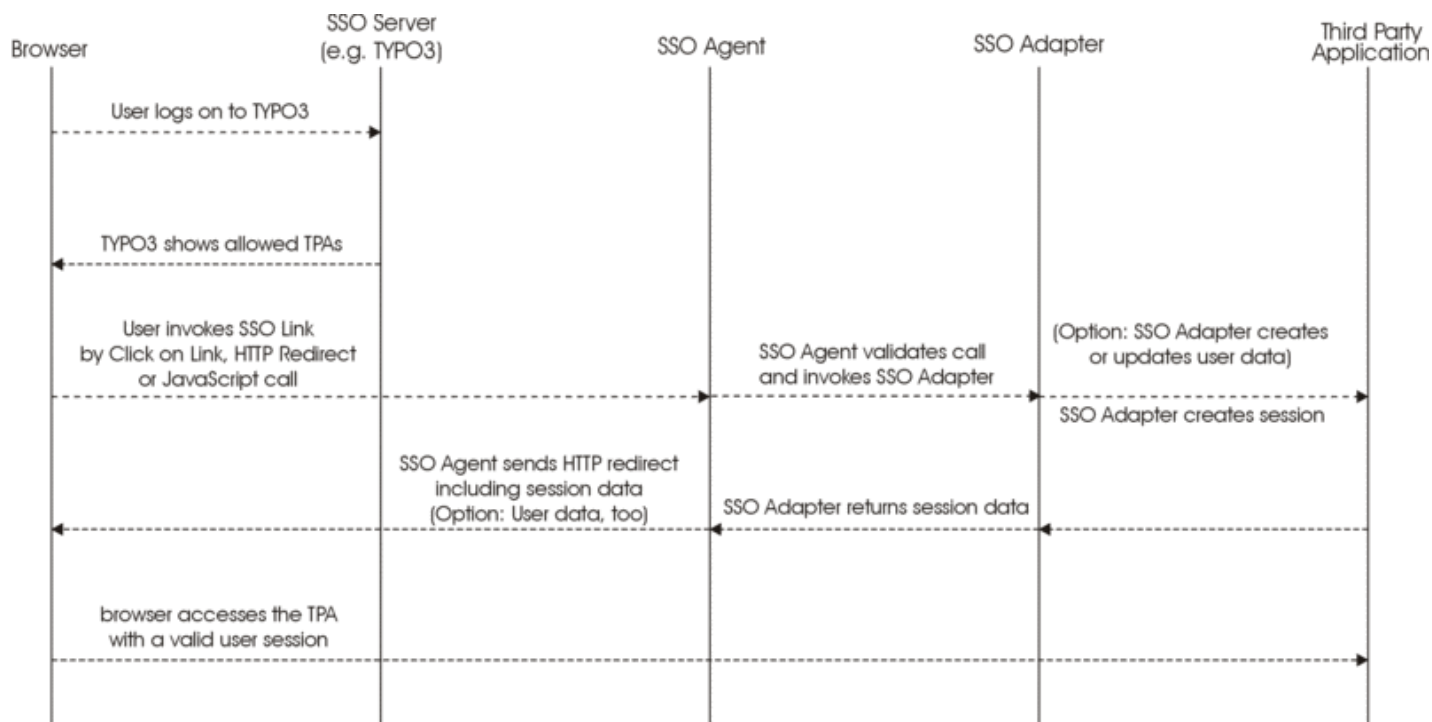


This image illustrates the following steps.

1. The user attempts to reach a hosted Google application, such as Gmail, Partner Start Pages (PSP) or another Google service.
2. Google generates a SAML authentication request. The SAML request is encoded and embedded into the URL for the partner's SSO service. The URL that the user is trying to reach is also embedded in the SSO URL.
3. Google sends a redirect to the user's browser. The redirect URL includes the encoded SAML authentication request that should be submitted to the partner's SSO service.
4. The partner decodes the SAML request and extracts the URL for both Google's ACS (Assertion Consumer Service) and the user's destination URL. The partner then authenticates the user. Partners could authenticate users by either asking for valid login credentials or by checking for valid session cookies.
5. The partner generates a SAML response that contains the authenticated user's username. In accordance with the SAML 2.0 specification, this response is digitally signed with the partner's public and private DSA/RSA keys.
6. The partner encodes the SAML response and the user's destination URL and returns that information to the user's browser. The partner provides a mechanism so that the browser can forward that information to Google's ACS. For example, the partner could embed the SAML response and destination URL in a form and provide a button that the user can click to submit the form to Google. The partner could also include JavaScript on the page that automatically submits the form to Google.
7. Google's ACS verifies the SAML response using the partner's public key. If the response is successfully verified, ACS redirects the user to the destination URL.
8. The user has been redirected to the destination URL and is logged in to Google Apps.

2.1.6. Typo3 SSO

Architecture



© 2005, net&works GmbH

TYPO3 Single Sign-On is signature-based and does NOT rely on a central reverse proxy, on server-to-server communication etc. but allows direct access to the TPA by securely passing a one-time-token to the browser (via URL). Thus, TPAs may be distributed across the net.

Basically, we find a 3-layer architecture:

- SSO Server (TYPO3 extension)

dynamically creates a link that includes the desired TPA, user name, and various security information.

- SSO Agent

The SSO Agent is located on each target system (the machine where the TPA lives), validates the incoming browser request, talks to the SSO Adapter, and gives back an HTTP redirect to the browser that points to the TPA itself.

- SSO Adapter

It is invoked by the SSO Agent. It creates a valid user session ("logs on the user") by application-specific means, and returns all information needed to the SSO Agent (in a defined format). This adapter is TPA-specific - this means that you need to find or develop an appropriate adapter for every TPA that you wish to integrate. It may be written in any language you favour.

SSO Server <-> SSO Agent

The SSO Agent is invoked via HTTP(S) access.

The SSO Server generates a link to the SSO Agent with the parameters

- **user** the user id that is to be logged on to the Third Party Application
- **tpa id** the id configured in the SSO Agent that identifies the desired TPA
- **expires** the absolute time until when this link may be used (seconds from 1970)
- **signature** the RSA/3DES signature (using the private key configured) of the above keys and values as represented in the URL string. The order of the parameters must not be changed.

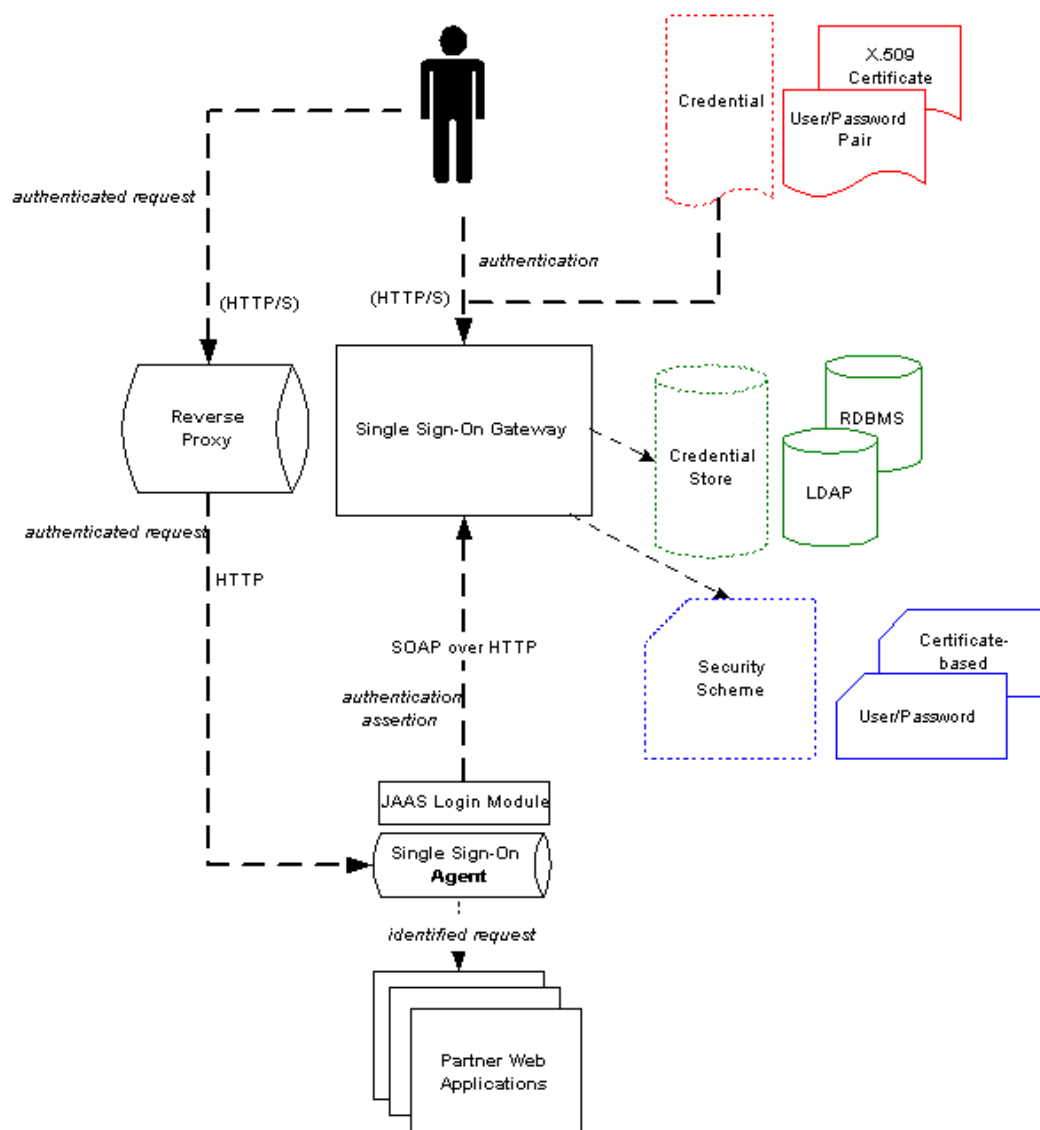
Example:

```

https://my.tpa/sigssso.php?user=mytestuser&tpa_id=MyOwnApp&expires=1076925657&signature=9c79040a0acf28a3512a4bea6609dd2a31cbbf7421b77817f97162d051f03ee76729e3f98d26690a5bc7967f6e6c38f0454fe71a9603c6a126d853f40b898721abf051a10b24100afd65458b95d2d5c37a4a5faa17c6be041caf028a9863049dcb15ae7c6cef47bfb171c47c50b3424fa5d6660207b57d5e2737a76c6eb8837187b317e0171029a4705c30dc4c7dcc0716c797ae712df82be315814ef5da5b21d8aeaf1537bdb16a6342bfbea64834853681d09461affa8ae09eb2388a104e1194141532cdc982ad9850ce6357065d8a1f5498f49ae7ba30865ed5d1d2760332528fd1455c025f4bd7702c83bd44dc839f5dc824d1582b024efb6a6a0aa3855
  
```

2.1.7. JOSSO

JOSSO, or Java Open Single Sign-On, is an open source J2EE-based SSO infrastructure aimed to provide a solution for centralized platform neutral user authentication.



3. Identity management

The Laws of Identity

The "Laws of Identity" are intended to codify a set of fundamental principles to which any universally adopted, sustainable identity architecture must conform. The Laws were proposed, debated, and refined through a long-running, open, and continuing dialogue on the Internet. Taken together, the Laws define the architecture of the identity metasystem.

They are:

1. User Control and Consent

Identity systems must only reveal information identifying a user with the user's consent.

2. Minimal Disclosure for a Constrained Use

The identity system must disclose the least identifying information possible, as this is the most stable, long-term solution.

3. Justifiable Parties

Identity systems must be designed so the disclosure of identifying information is limited to parties having a necessary and justifiable place in a given identity relationship.

4. Directed Identity

A universal identity system must support both "omni-directional" identifiers for use by public entities and "uni-directional" identifiers for use by private entities, thus facilitating discovery while preventing unnecessary release of correlation handles.

5. Pluralism of Operators and Technologies

A universal identity solution must utilize and enable the interoperation of multiple identity technologies run by

multiple identity providers.

6. Human Integration

Identity systems must define the human user to be a component of the distributed system, integrated through unambiguous human-machine communication mechanisms offering protection against identity attacks.

7. Consistent Experience Across Contexts

The unifying identity metasystem must guarantee its users a simple, consistent experience while enabling separation of contexts through multiple operators and technologies.

© Kim Cameron

4. Bibliographic references

- [Authentication \(Wikipedia\)](#)
- [The Failure of Two-Factor Authentication](#)
- [Kerberos \(Wikipedia\)](#)
- [Needham-Schroeder \(Wikipedia\)](#)
- [Kerberos page at MIT](#)
- [Designing an Authentication System: a Dialogue in Four Scenes](#)
- [How the Kerberos Version 5 Authentication Protocol Works in Windows Servers](#)
- [Single sign on \(Wikipedia\)](#)
- [SSO standardisation \(The OpenGroup\)](#)
- [Single Sign On \(AuthenticationWorld.com\)](#)
- [Windows Live ID \(Wikipedia\)](#)
- [Windows Live ID Service](#)
- [WS-Trust](#)
- [WS-Security \(Wikipedia\)](#)
- [Liberty Alliance \(Wikipedia\)](#)
- [Liberty Alliance](#)
- [SAML \(Wikipedia\)](#)
- [SAML Single Sign-On \(SSO\) Service for Google Apps](#)
- [Single Sign-On for TYPO3 \(and others\)](#)
- [TYPO3 Single Sign-On whitepaper](#)
- [Signature-Based Single Sign-On Framework](#)
- [OSSO \(Java Open Single Sign-On\)](#)
- [Laws of Identity](#)
- [identityblog](#)