e i g
Ecole d'ingénieurs
de Genève

Hes·so
Haute Ecole Spécialisée
de Suisse Occidentale

Web Services

ADFS

Etudiant : Pellarin Anthony

Professeur : Litzistorf Gérald

En collaboration avec : Sogeti

Date du travail : 15.10.2007

Table des matières :

1. Introduction

La seconde partie de mon travail de diplôme qui à été proposé par la société Sogeti à été l'étude des Web Services, la sécurité associée à ceux-ci ainsi que l'étude d'une méthode d'authentification basée sur la fédération d'identité, Active Directory Federation Services (ADFS).

Les Web Services sont de plus en plus présents aujourd'hui. Ils permettent entre autres l'échange de données à travers Internet indépendamment des langages et des plates-formes utilisées grâce à un ensemble de protocoles standardisés comme SOAP, WSDL ou UDDI.

Les Web Services ont étés conçus pour faciliter l'échange de données ainsi que l'accès aux applications au sein des entreprises mais également entre les entreprises.

Plusieurs questions de sécurité se posent alors. En effet les Web Services introduisent un certain nombre de vulnérabilités dont il faut faire attention. Certaines failles de sécurité doivent être corrigées, l'injection de code XML corrompu volontaire ou involontaire doit pouvoir être évité et certaines règles de base en matière de sécurité doivent être imposées.

La sécurité est généralement un point sensible dans les entreprises pour diverses raisons : Peu de budget alloué aux aspects sécurité des Web Services, peu d'études sont effectuées sur le sujet, ou encore, pas de moyen pour la recherche de solutions.

Les premiers chapitres seront donc consacrés aux aspects de sécurités liés aux Web Services et aux différentes failles que l'on peut trouver et exploiter.

La suite du projet est consacrée à l'étude de l'architecture d'authentification ADFS qui permet à une entreprise de partager une application avec des partenaires sans devoir inclure dans son annuaire les utilisateurs qui vont se connecter à l'application et sans que les utilisateurs partenaires aient à se réauthentifier en utilisant un mot de passe supplémentaire. En effet cette technologie permet à une entreprise d'étendre son annuaire AD au travers d'internet d'une manière sécurisé par le biais de jetons de sécurité pour être exploité par une application.

Pour mettre en œuvre cette technologie, divers scénarios ont été mis en place pour montrer les différentes applications pouvant utiliser une authentification ADFS.

- Un premier scénario décrira l'accès à une application .NET ainsi que l'exploitation du jeton de sécurité pour attribuer les diverses autorisations nécessaires.
- Un second scénario montrera comment l'application Windows SharePoint Services peut être modifiée pour authentifier des utilisateurs en utilisant ADFS.
- Le troisième scénario est basé sur l'accès à un Web Services avec une authentification ADFS.
- Finalement, le dernier scénario présentera l'interopérabilité offerte par WS-Federation avec l'accès à une application située sur un serveur Linux avec le mécanisme d'authentification ADFS

Le **chapitre 2** présente les Web Services, les différents standards utilisés ainsi que l'architecture des Web Services. Il énumère aussi les différentes sociétés mettant à disposition une API servant à programmer une application Web.

Le **chapitre 3** traite des aspects sécurité des Web Services. Il présente les différents failles potentielle, les méthodes pour essayer de s'en protéger et il introduit le protocole de sécurité WS-Security.

Le **chapitre 4** est consacré à Active Directory Federation Services. Il traite de l'architecture utilisée, des différents acteurs présents, des jetons de sécurité et présente les différents scénarios pratiques mis en œuvre.

Temps consacré aux différents points de l'énoncé :

Etude des Web Services, du protocole WS-Security et implémentation pratique

- 1 semaine

Etude de l'architecture ADFS

- 1 semaine

Mis en place des divers scénarios

- 4 semaines

2. Web Services

Le Web à rencontré un grand succès en permettant des interactions simples entre l'utilisateur et l'ordinateur au niveau d'Internet. Le principal facteur du succès de HTTP et de HTML réside dans leur relative simplicité.

Les services Web reprennent la plupart des idées et des principes du Web, et les appliquent à des interactions entre ordinateurs. Comme pour le World Wide Web, les services Web communiquent via un ensemble de protocoles fondamentaux.

Les services Web communiquent par messages, ils supposent donc que SOAP est employé dans les couches basses de la pile des protocoles afin d'isoler le transfert des messages des détails de la couche transport. Cette approche constitue une base solide pour atteindre l'interopérabilité des services entre diverses plates-formes de développement, et elle permet des structures de communication plus riches. Les services Web s'appuient généralement sur HTTP au niveau du transport des messages. De cette manière, de nombreux services Web ont pu se développer en exploitant des technologies existant déjà sur le Web.

2.1. Standards des Web Services

Les services Web sont donc une technologie permettant à des applications de dialoguer via Internet, par l'échange de messages fondés sur des standards, et ceci indépendamment des plates-formes et des langages sur lesquelles elles reposent. Il y'a deux acteurs principaux dans l'utilisation de services Web :

- Le fournisseur de service
- Le demandeur de service

Un troisième acteur peut éventuellement venir s'ajouter à ce couple, l'annuaire de service permettant de répertorier l'ensemble des fournisseurs de services.

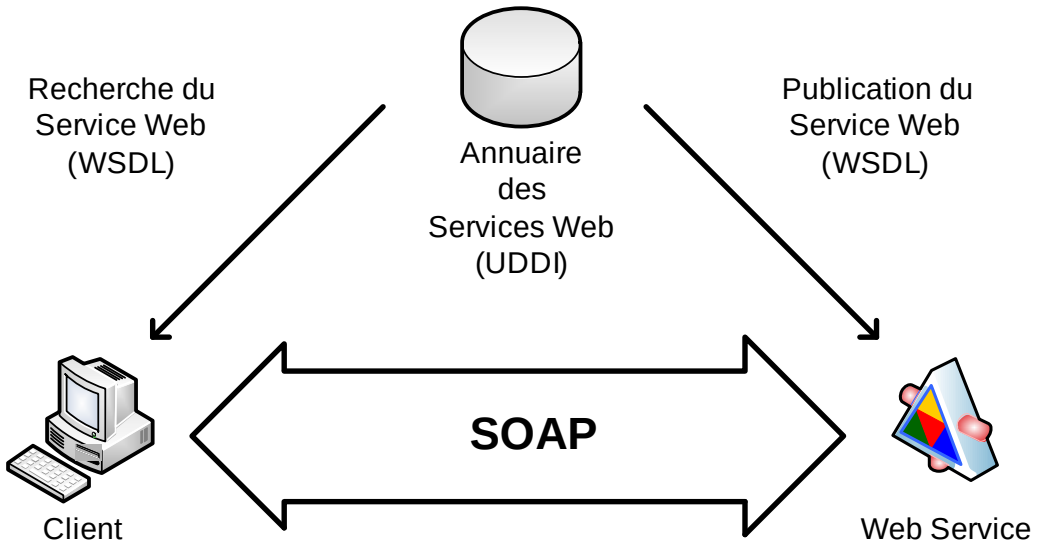


Figure 2.1 Acteurs des services Web

Les services Web s'appuient sur un ensemble de protocoles standards.

La communication s'effectue grâce aux protocoles standards d'Internet : HTTP, HTTPS,…

Les services Web se fondent sur XML[1] qui est un langage de bannières permettant d'écrire des contenus organisés de manière hiérarchique. Le principal intérêt d'XML est que ce format de document permet de transmettre l'information, mais aussi les métadonnées qui indiquent sa signification, et la structure de l'information à échanger.

Les protocoles mis en œuvre pour un échange basique entre un demandeur et un fournisseur sont illustrés par la figure 2.2.

[1] http://www.w3.org/TR/xml11/

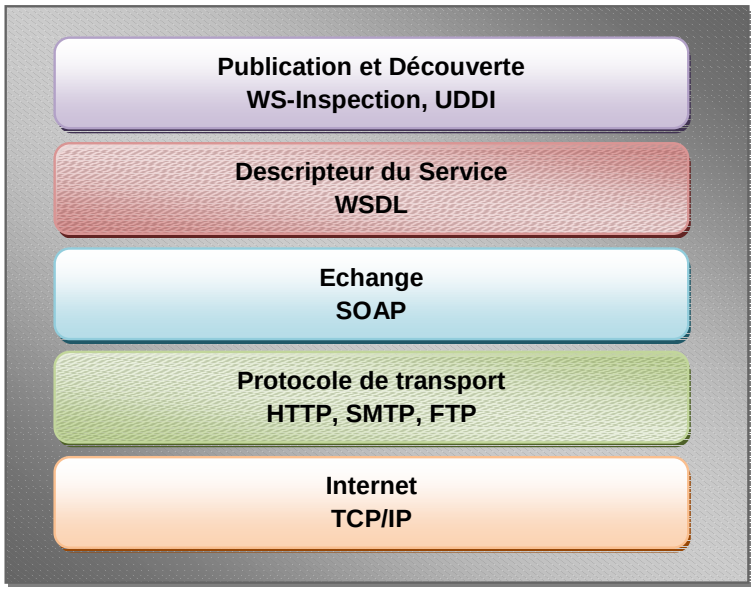


Figure 2.2 Normes de bases

La totalité des acteurs de l'industrie des services Web adhèrent au socle technique des services Web : les protocoles UDDI, WSDL et SOAP. Ces normes régissent l'interopérabilité des services Web et permettent de faire interopérer des systèmes d'information hétérogènes :

- **UDDI**[2] définit les formats pour crées des catalogues pour publier et rechercher des services Web.
- **WSDL**[3] permet de spécifier le format des messages, les protocoles qui doivent être utilisés et la localisation des différentes machines qui mettent en œuvre un service Web.
- **SOAP**[4] permet la normalisation des échanges de données.

[2] http://uddi.org/pubs/uddi_v3.htm

[3] http://www.w3.org/TR/wsdl

[4] http://www.w3.org/TR/soap12-part1/

2.2. Architecture des services Web

Les services Web ont rapidement étés adoptés par l'industrie des technologies de l'information. Face au succès de cette vague technologique, les éditeurs se doivent d'intégrer un support des services Web au sein de leurs offres produits. L'intérêt pour les services Web a donc su fédérer les principaux acteurs économiques du marché mondial, comme Microsoft, IBM, et beaucoup d'autres.

Cependant les services Web sont appelés à évoluer. Pour offrir un contrôle sur la cohérence de ces évolutions, les différents acteurs concernés se doivent de participer à la définition des futures orientations de ces services Web. Ils ont donc décidé de travailler ensemble à l'élaboration des standards auxquels chacun devrait se plier pour que les services Web soient pleinement interopérables.

Les protocoles déjà existants ainsi que les évolutions sont organisées dans une architecture nommée ***Web Service Architecture*** (WSA)[5] :

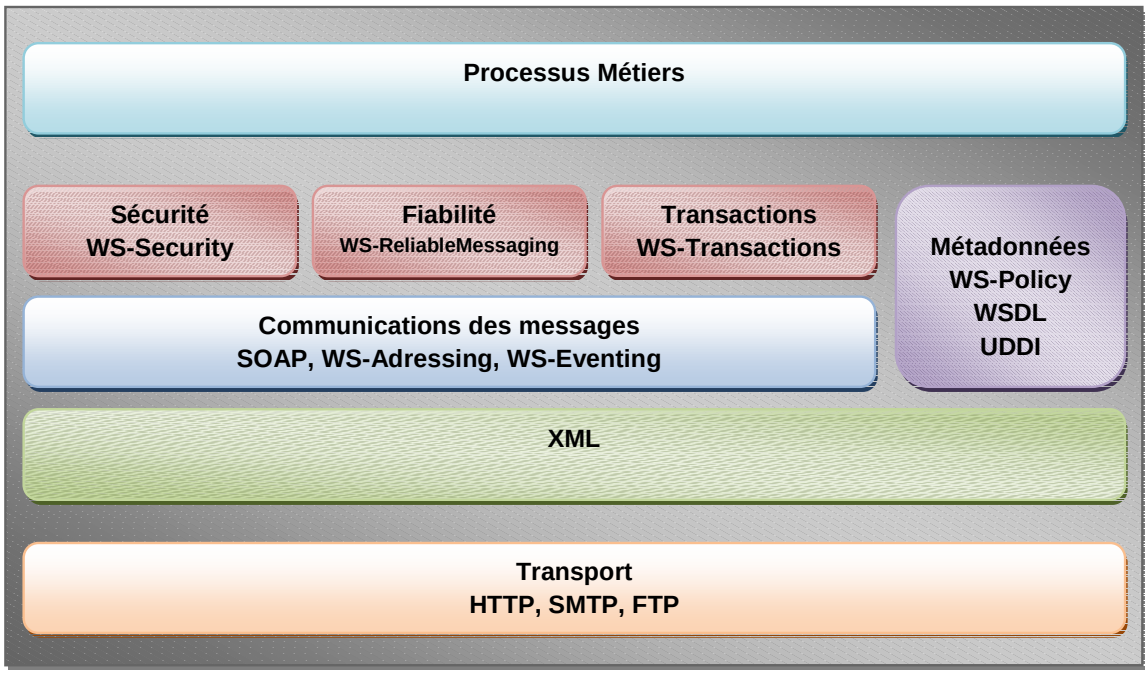


Figure 2.3 *Web Service Architecture*

[5] http://www.w3.org/TR/ws-arch/

Le socle de ce *framework* de services génériques est constitué par la couche de transport qui propose différents protocole standards assurant les échanges de données (HTTP, HTTPS, SMTP,…). Cette couche n'est plus à remettre en cause pour l'étude des services Web.

Elle est complétée par la spécification XML, standard universel de manipulation des informations, dont la syntaxe est le fondement de la définition des autres protocoles Web services.

La couche de communication des messages propose différents mécanismes liés à l'acheminement des messages (format de communication des messages, adressage, routage,…).

Les blocs suivants concernent les domaines sur lesquels porte la majeure partie des évolutions actuelles. Ces spécifications ont pour objectif d'offrir aux services Web le support de mécanismes complexes de sécurité, de garantie de livraison, de transaction, de description, etc. Ces spécifications peuvent encore évoluer, tandis que de nouvelles spécifications se créent et que leur complexité et leur couplage s'intensifient.

Enfin la dernière couche, située au sommet de l'organisation WSA, permet de définir la gestion des processus métiers construits avec des services Web. C'est notamment à ce stade que l'on peut voir émerger la véritable notion de service, pièce maîtresse des architectures orientées services.

2.3. Web Services aujourd'hui

Aujourd'hui, des entreprises mettent à disposition des codes sources (SDK) pour le développement de web services et ainsi pouvoir utiliser les différents services proposés par cette entreprise.

Les sociétés mettent par exemple à disposition un fichier WSDL qui décrit les variables d'entrées et de sortie de l'application et donc sous quel forme doivent être passés les paramètres.

```
- <definitions name="GoogleSearch" targetNamespace="urn:GoogleSearch" xmlns:typens="urn:GoogleSearch"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:soapenc="http://schemas.xmlsoap.org/
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" xmlns="http://schemas.xmlsoap.org/wsdl/">
    <!--  Types for search - result elements, directory categories  -->
  - <types>
    - <xsd:schema xmlns="http://www.w3.org/2001/XMLSchema" targetNamespace="urn:GoogleSearch">
      - <xsd:complexType name="GoogleSearchResult">
        - <xsd:all>
            <xsd:element name="documentFiltering" type="xsd:boolean" />
            <xsd:element name="searchComments" type="xsd:string" />
            <xsd:element name="estimatedTotalResultsCount" type="xsd:int" />
            <xsd:element name="estimateIsExact" type="xsd:boolean" />
            <xsd:element name="resultElements" type="typens:ResultElementArray" />
            <xsd:element name="searchQuery" type="xsd:string" />
            <xsd:element name="startIndex" type="xsd:int" />
            <xsd:element name="endIndex" type="xsd:int" />
            <xsd:element name="searchTips" type="xsd:string" />
            <xsd:element name="directoryCategories" type="typens:DirectoryCategoryArray" />
            <xsd:element name="searchTime" type="xsd:double" />
          </xsd:all>
        </xsd:complexType>
```

Figure 2.4 WSDL Google

Dans cet extrait d'un fichier WSDL mis à disposition par Google pour ses applications Web, on voit donc les différentes variables qui sont utilisées et quel est leur type.

Grâce à ce fichier on peut donc savoir quel est le type des variables que l'on doit envoyer dans la requête SOAP.

La société **Michelin**[6] met à disposition un SDK pour la programmation d'un service web qui va pouvoir utiliser des services telles que :

- La génération de cartes
- Le calcul d'itinéraires et de distances entre deux points

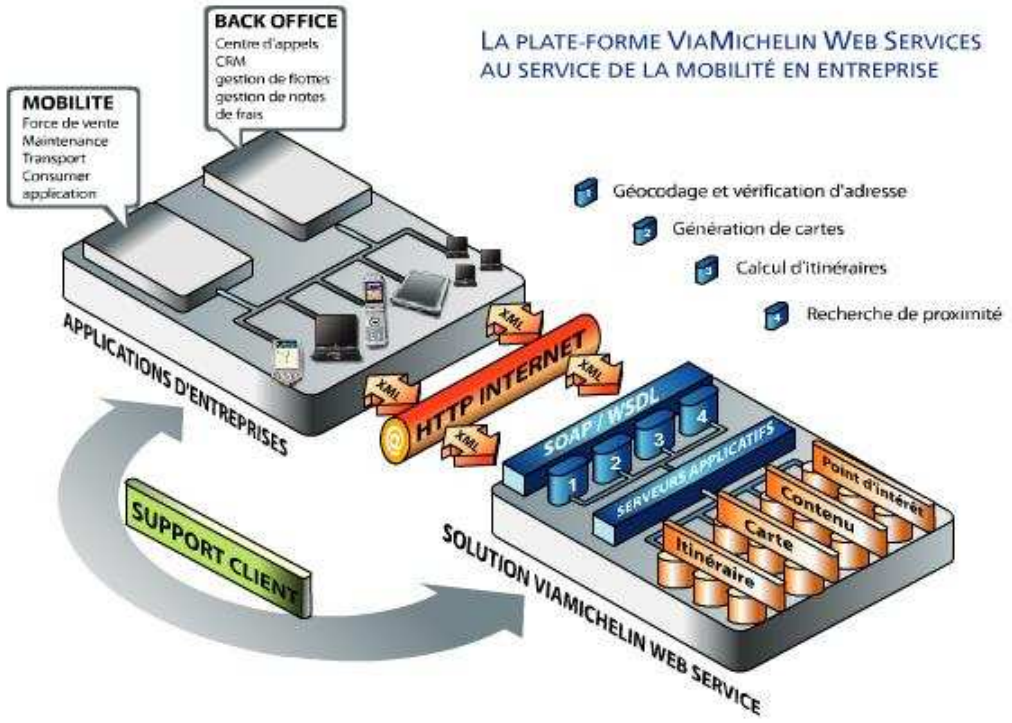


Figure 2.5 Plate-forme ViaMichelin Web Services

[6] http://business.viamichelin.co.uk/b2b/web_services.html

Le site de vente en ligne **Amazon**[7] propose plusieurs services Web que l'on peut implémenter comme par exemple :

- Un service E-Commerce
- Un service de paiement en ligne
- Intégration du moteur de recherche Alexa

La société **Google**[8], réputée pour son puissant moteur de recherche, met aussi à disposition un service Web permettant d'intégrer le moteur de recherche dans n'importe quelle application .NET. Le site ne propose plus d'implémentation utilisant SOAP mais offre un SDK utilisant AJAX pour par exemple :

- Intégrer le moteur de recherche dans une application
- Utiliser les résultats de recherches

[7] http://www.amazon.com/AWS-home-page-Money/b/ref=sc_iw_l_0?ie=UTF8&node=3435361&no=3435361&me=A36L942TSJ2AJA

[8] http://code.google.com/

MapPoint[9], qui est un logiciel de Microsoft ayant pour but d'offrir les fonctions d'un de GPS met à disposition un kit de développement qui offre par exemple les fonctions suivantes :

- Calcul d'itinéraire
- Recherche de rues, bâtiments,…
- Recherche d'accidents en temps réel

Figure 2.6 MapPoint

[9] http://www.microsoft.com/mappoint/products/webservice/default.mspx

2.4. Bibliographie et Liens

- Valérie Monfort, Stéphane Goudeau. (2004).

  Web Services et intéropérabilité des SI. Dunod

  Chapitre 1 – Services Web et architectures orientées services

- **Introduction à l'architecture de services Web et ses spécifications WS-***
- http://msdn2.microsoft.com/fr-fr/library/887bf053-f951-4ede-bb68-6920c4304133.aspx

3. Sécurité des Web Services

Les services web permettent donc aux applications d'interopérer en se fondant sur des standards internet tels que XML, SOAP, WSDL, UDDI… Ces protocoles permettent dès aujourd'hui de mettre en places des applications mais de nombreuses évolutions restent à apporter pour offrir la prise ne compte de critère de qualité de service.

La sécurisation d'une infrastructure basée sur une architecture SOA, s'avère beaucoup plus complexe que celle des environnements traditionnels. En effet les services proviennent de plates-formes différentes impliquant la nécessité de pouvoir proposer un modèle d'abstraction qui est indépendant de la plate-forme utilisée. D'autre part, le modèle proposé doit pouvoir s'appuyer sur un modèle existant en termes de sécurité.

Les futures spécifications de sécurité des services Web servant à résoudre ces problèmes sont en cours de définition. Ce chapitre présente la sécurité des services Web et le protocole WS-Security.

3.1. Types de menaces et solutions

Les applications qu'elles soient ou non constituées de services Web, doivent faire face à un certain nombre de menaces :

- **Spoofing identity** : un utilisateur non autorisé usurpe l'identité d'un utilisateur légitime.

- **Tampering of data** : un utilisateur détruit ou modifie les informations sans autorisation.

- **Repudiability** : la possibilité qu'un utilisateur puisse nier avoir effectué telle ou telle opération.

- **Information disclosure** : des données confidentielles sont rendues visibles à des utilisateurs non autorisés

- **Denial of service** : l'application est rendue indisponible

- **Elevation of privilege** : un utilisateur dispose d'un niveau d'accès supérieur à celui qui devrait lui être accordé.

Pour se prémunir de ces menaces, différentes techniques sont mise en œuvre :

- **Authentification** : identification des applications clientes d'un service Web. Les mécanismes d'authentification permettent de se prémunir contre l'usurpation d'identité (signature).

- **Autorisation** : définition des droits d'un client authentifié. Les mécanismes de gestion des autorisations permettent d'éviter l'altération de données et la diffusion d'informations sensibles.

- **Identité** : il est parfois nécessaire de véhiculer le contexte de sécurité de l'utilisateur via de multiples ressources du système d'information.

- **Sécurité des communications** : il faut faire en sorte que les messages circulant sur le réseau restent privées (chiffrement) et non altérés (signature du message).

- **Audit** : enregistrement des actions de l'application cliente, qu'elles soient autorisées ou non, pour garantir la non-répudiation.

L'architecture de spécification de sécurité est représentée par un diagramme à niveaux :

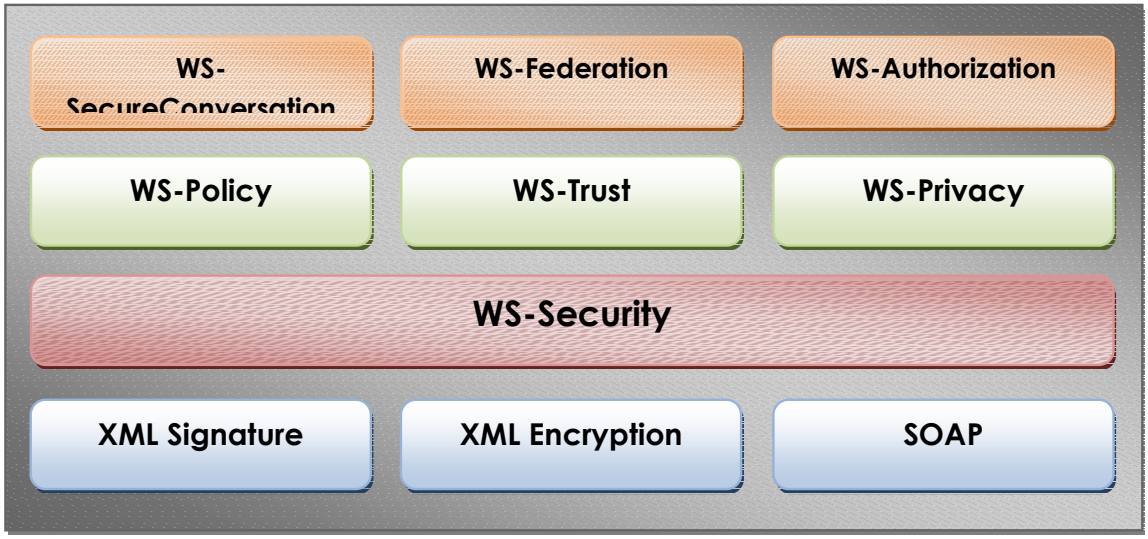


Figure 3.1 Architecture des spécifications de l'infrastructure de sécurité pour les services Web

Les bases de l'architecture des spécifications sont les recommandations W3C *XML Signature* et *XML Encryption* et, bien évidemment, le protocole SOAP. C'est une architecture des spécifications construite sur trois strates :

- La première strate comprend une spécification de sécurité (***WS-Security***), qui met en œuvre l'intégrité et la confidentialité du message et fournit le mécanisme d'association de jetons de sécurité au message.

- La deuxième strate, intermédiaire, comprend les spécifications ***WS-Policy*** (langage de définition d'engagement de sécurité), ***WS-Trust*** (un *framework* pour construire des modèles de confiance) et ***WS-Privacy*** (un modèle pour les assertions sur les préférences en termes de politique de discrétion et de confidentialité).

- La troisième strate comprend ***WS-SecureConversation*** (spécification de protocole d'authentification réciproque et d'établissement des contextes de sécurité entre applications), ***WS-Federation*** (mécanisme de fédération d'espaces de confiance hétérogène) et ***WS-Authorization*** (spécification des protocoles d'autorisation).

3.2. Vulnérabilités

Il existe plusieurs sortes de vulnérabilité[10] que l'on peut rencontrer dans les services Web. La sécurité dans les services Web est quelque chose qui n'est pas encore bien en place et qui n'occupe pas encore une part importante des applications actuelles.

Aujourd'hui, grâce à l'utilisation du protocole SSL par exemple ou encore l'utilisation de signature XML, le risque de vol d'identité, d'interception des données ou encore l'atteinte à l'intégrité et à la confidentialité des données est donc diminué.

Mais ces mesures de sécurité n'apportent rien pour la minimisation des risques d'intrusions dans les applications. Une application dont le contrôle a été acquis pourra être corrompue.

3.2.1. Cross Site Scripting (XSS)

Le *Cross Site Scripting*[11] est une faille de sécurité que l'on rencontre dans les applications Web qui permet à un utilisateur malveillant d'afficher des pages web avec du code douteux.

Le principe est d'injecter des données au site Web par l'intermédiaire de paramètres par exemple et si les données sont transmise au site Web sans avoir été au préalable contrôlées, alors on peut en déduire qu'une faille de ce type est présente.

L'exploitation de ce type de faille permet à un intrus de réaliser les opérations suivantes :

- Affichage d'un contenu non-interne au site (publicités, faux articles,…)
- Redirection de l'utilisateur de manière transparente
- Vol d'informations
- Actions sur le site à l'insu de la victime
- Plantage de la page ou du navigateur

Pour se prémunir de ce type de faille, il faut donc bien vérifier les informations qui sont transmise au site de façon à contrôler qu'elles ne soient pas potentiellement dangereuses.

[10] http://www.securityfocus.com/vulnerabilities

[11] http://www.owasp.org/index.php/Alternate_XSS_Syntax

3.2.2. Injection XML

Le but de l'attaque par injection XML[12] va donc être d'envoyer au serveur des données, simplement en rentrant des informations d'identification par exemple, mais en envoyant des caractères spéciaux qui pourrait ne pas être pris en charge par l'application et donc de porter atteinte à celle-ci.

Par exemple[13] une application ou l'on rentrer ses informations à savoir :

Username : Alice
Password : Secret

Ces informations vont produire une requête :

http://www.example.com/addUser.php?username=Alice&password=Secret

Qui finalement va produire le code XML suivant :

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<users>
    <user>
        <username>Alice</username>
        <password>Secret</password>
    </user>
</users>
```

Pour tenter d'exploiter cette faille, on va donc tenter de rentrer des informations composée par exemple de métacaractères comme **<** ou **'** ou **"**.

La requête pourrait donc être formulée ainsi :

Username : Charly<
Password : hacker

[12] http://www.owasp.org/index.php/XPATH_Injection

[13] http://www.owasp.org/index.php/Testing_for_XML_Injection

Le code XML produit à ce moment la serait :

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<users>
	<user>
		<username>Charly<</username>
		<password>Hacker</password>
	</user>
</users>
```

Ce qui produit donc un code XML non-valide, et empêche la validation du code.

Un autre exemple d'attaque est de rajouter une information que l'on met soi-même dans la requête en spécifiant par exemple manuellement son ID alors qu'en temps normal il est génère automatiquement.

L'utilisateur rentre alors les informations de cette manière :

Username : Charly</username><ID>0</ID><username>Charly

La requête XML crée sera alors de cette forme :

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<users>
	<user>
		<username> Charly</username><ID>0</ID><username>Charly
</username>
		<password>Hacker</password>
	</user>
</users>
```

Charly aura donc un ID de 0 qui pourrait par exemple représenter les administrateurs.

3.2.3. Attaques SOAP

Une requête SOAP[14] comme on l'a vu, est décrite par un fichier WSDL qui indique la structure de la requête et ensuite, le message SOAP se compose d'un en-tête et d'un corps.

Lorsqu'on envoie une requête SOAP, on envoie des paramètres à l'application et celle-ci nous répond en nous renvoyant un message SOAP contenant les réponses aux requêtes.

Par exemple, une entreprise avec un système de gestion des stocks. On interroge la base de données par une requête SOAP en indiquant par exemple la référence de l'objet à recherche. L'application nous renvoie ensuite des informations comme par exemple le nom de l'objet.

Un type d'attaque serait donc par exemple de mettre un caractère spécial dans le champ de la recherche de l'article.

La requête SOAP se présente donc sous cette forme si on recherche un article ayant comme nom : **'**

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
- <SOAP-ENV:Envelope xmlns:SOAPSDK1="http://www.w3.org/2001/XMLSchema"
    xmlns:SOAPSDK2="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:SOAPSDK3="http://schemas.xmlsoap.org/soap/encoding/" xmlns:SOAP-
    ENV="http://schemas.xmlsoap.org/soap/envelope/">
  - <SOAP-ENV:Body>
    - <SOAPSDK4:GetProductInformationByName
        xmlns:SOAPSDK4="http://sfaustlap/ProductInfo/">
        <SOAPSDK4:name>'</SOAPSDK4:name>
        <SOAPSDK4:uid>551-457-4487</SOAPSDK4:uid>
        <SOAPSDK4:password>123456</SOAPSDK4:password>
      </SOAPSDK4:GetProductInformationByName>
    </SOAP-ENV:Body>
  </SOAP-ENV:Envelope>
```

On remarque donc le champ **name** :

<SOAPSDK4:name>'</SOAPSDK4:name>

[14] http://www.spidynamics.com/whitepapers/SOAP_Web_Security.pdf

La réponse renvoyée par l'application nous indique une erreur en indiquant quelques informations quant au fonctionnement interne du service Web:

```
<?xml version="1.0" encoding="utf-8" ?>
- <soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  - <soap:Body>
    - <soap:Fault>
        <faultcode>soap:Server</faultcode>
        <faultstring>System.Web.Services.Protocols.SoapException: Server was
          unable to process request. ---> System.Data.OleDb.OleDbException:
          Syntax error (missing operator) in query expression 'productname like '''
          and providerid = '551-457-4487''. at
          System.Data.OleDb.OleDbCommand.ExecuteCommandTextErrorHandling
          (Int32 hr) at
          System.Data.OleDb.OleDbCommand.ExecuteCommandTextForSingleResult
          (tagDBPARAMS dbParams, Object& executeResult) at
          System.Data.OleDb.OleDbCommand.ExecuteCommandText(Object&
          executeResult) at System.Data.OleDb.OleDbCommand.ExecuteCommand
          (CommandBehavior behavior, Object& executeResult) at
          System.Data.OleDb.OleDbCommand.ExecuteReaderInternal
          (CommandBehavior behavior, String method) at
          System.Data.OleDb.OleDbCommand.ExecuteReader(CommandBehavior
          behavior) at System.Data.OleDb.OleDbCommand.ExecuteReader() at
          ProductInfo.ProductDBAccess.GetProductInformation(String productName,
          String uid, String password) at
          ProductInfo.ProducInfo.GetProductInformationByName(String name, String
          uid, String password) --- End of inner exception stack trace ---</faultstring>
        <detail />
      </soap:Fault>
    </soap:Body>
  </soap:Envelope>
```

Maintenant, on peut essayer d'utiliser d'autres caractères comme des *wildcards* ou encore de mettre des conditions dans les requêtes.

Ici, pour connaitre les informations sur un Provider, on indique l'uid et le password. Ne connaissant pas le password, on va entrer à la place une condition qui se révèle être toujours true :

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
- <SOAP-ENV:Envelope xmlns:SOAPSDK1="http://www.w3.org/2001/XMLSchema"
    xmlns:SOAPSDK2="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:SOAPSDK3="http://schemas.xmlsoap.org/soap/encoding/" xmlns:SOAP-
    ENV="http://schemas.xmlsoap.org/soap/envelope/">
  - <SOAP-ENV:Body>
    - <SOAPSDK4:GetProviderInfo
        xmlns:SOAPSDK4="http://sfaustlap/ProductInfo/">
        <SOAPSDK4:uid>551-457-4487</SOAPSDK4:uid>
        <SOAPSDK4:password>' or 1=1 or password = '</SOAPSDK4:password>
      </SOAPSDK4:GetProviderInfo>
    </SOAP-ENV:Body>
  </SOAP-ENV:Envelope>
```

L'instruction remplaçant le password :

<SOAPSDK4:password>' or 1=1 or password = '</SOAPSDK4:password>

La réponse que l'on reçoit en retour :

```
<?xml version="1.0" encoding="utf-8" ?>
- <soap:Envelope
    xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  - <soap:Body>
    - <GetProviderInfoResponse
        xmlns="http://sfaustlap/ProductInfo/">
      - <GetProviderInfoResult>
          <providerid>551-457-4487</providerid>
          <name>Mom and Pop Inc</name>
          <password>123456</password>
          <discount>20</discount>
        </GetProviderInfoResult>
      </GetProviderInfoResponse>
    </soap:Body>
  </soap:Envelope>
```

On constate donc que les informations sur le Provider dont l'uid correspondait à celui que l'on a rentré dans la requête nous sont renvoyées en nous indiquant entre autre le password du Provider en question.

Pour se protéger de ce type d'attaque, il convient donc de mettre en place certaines règles au niveau de la sécurité.

Toutes les donnes **entrantes** que l'application reçoit par des requêtes SOAP doivent être vérifiée pour contrôler qu'aucun caractère non autorise est envoyé. In ne faut pas faire aveuglement confiance aux informations XML que l'on reçoit. Il faut contrôler les chaines de caractères pour pas qu'un "\" ou un "' ne puisse être incorpore à la requête et compromette la réponse.

Les données **sortantes** aussi doivent être vérifiées. Il faut s'assurer que les réponses renvoyées au client sont corrects, valider le format des réponses et utiliser des pages d'erreurs génériques pour ne pas donner trop d'informations au client qui pourrait ensuite s'en servir d'une manière malveillante.

3.3. Protocole WS-Security

3.3.1. Pourquoi WS-Security ?

Il y'a aujourd'hui des applications en production mettant en œuvre des services Web, sans mettre en application ce protocole. Ces applications sont évidemment sécurisées en exploitant les protocoles de transport standards :

- **SSL** (*Secure Socket Layer*) : protocole conçu pour assurer la confidentialité des échanges entre un client et un serveur Web généralement sur le protocole HTTP.

- **IPSec** (*IP Security Protocol*) : protocole de sécurité réseau faisant intégrante de la pile IP utilisé entre deux serveurs ou deux « passerelles de sécurité » ou entre un serveur et une passerelle afin de sécuriser les sessions en offrant des mécanismes d'authentification, d'intégrité et de confidentialité des données.

Ce modèle de sécurité et un modèle simple, adapté aux solutions pour lesquelles les mécanismes de transports et la configuration des points de terminaisons sont complètement maitrisés.

Quelles sont les limites d'une telle approche pour garantir l'intégrité et la confidentialité des messages ?

HTTPS permet le chiffrement et la signature du message, ainsi que l'authentification du client, ce qui permet de garantir la non-répudiation, la confidentialité, et l'intégrité des informations échangées. Mais dans certains contextes, cela se révèle insuffisant pour plusieurs raisons :

- Le protocole SOAP n'est pas exclusivement lié au protocole HTTP.

- Il peut résulter une dépendance vis-à-vis de la plate-forme et du fournisseur de service de sécurité (NTLM, Kerberos, etc.).

- La topologie d'une solution distribuée fondée sur les services Web peut inclure de nombreux périphériques ou systèmes intermédiaires. Il est donc primordial de pouvoir sécuriser les échanges entre applications et services transitant entre ces nœuds intermédiaires. Dans ce type de configuration, le protocole HTTPS ne gère qu'une sécurité de point à point, pas debout en bout. Comment faire alors pour maintenir le contexte de sécurité sur la globalité de la chaîne ?

Dans ce modèle (Figure 3.1), la sécurité n'est garantie que par la plate-forme et le transport :

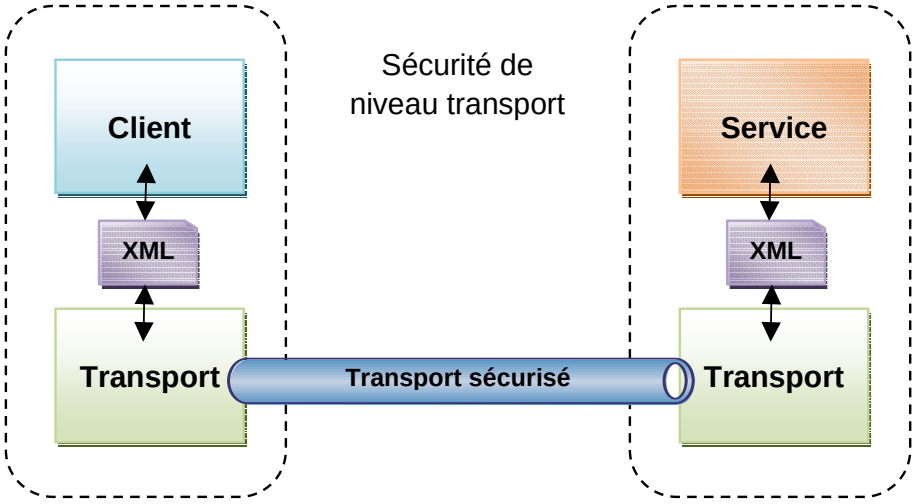


Figure 3.2 Sécurité de point à point

3.3.2. WS-Security

Adresser la problématique de sécurité à un niveau indépendant de la couche transport permet de pallier à ces limites. En se fondant sur une sécurité de niveau messages, WS-Security[15] permet de sécuriser une solution mettant en œuvre une multiplicité de plates-formes, sans nécessité d'avoir la configuration des différents nœuds, extrémité ou intermédiaire, intervenant dans l'échange et en offrant des compléments en termes de chiffrement et de non-répudiation.

Dans ce modèle de sécurité, les messages XML contiennent les informations d'identification, les signatures numériques qui peuvent être chiffrées.

La figure 3.2 illustre le fonctionnement d'une solution mettant en œuvre cette spécification :

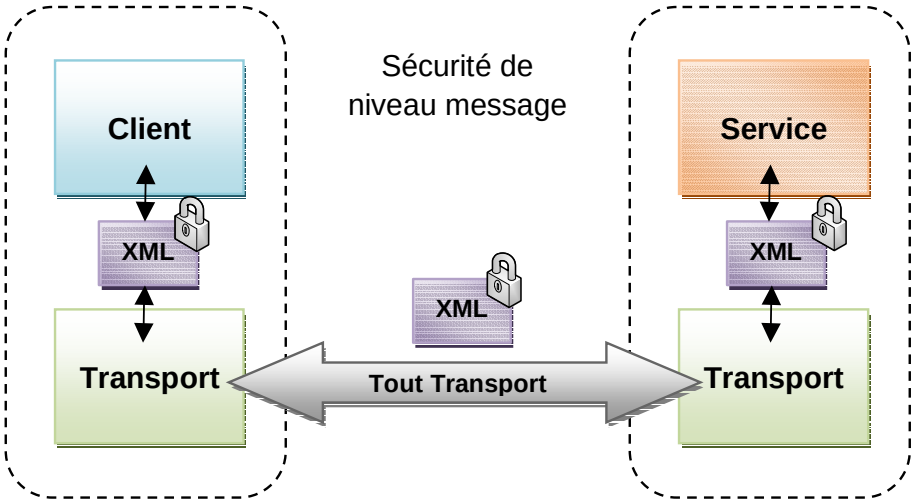


Figure 3.3 Sécurité au niveau message

[15] http://msdn2.microsoft.com/en-us/library/ms951257.aspx

La spécification WS-Security définit un ensemble d'extension SOAP standards qui implémente la sécurité (intégrité et confidentialité). Plutôt que de proposer un nouveau Framework de sécurité, WS-Security s'inspire des mécanismes existants permettant la mise en place d'une infrastructure sécurisée et propose trois mécanismes constituant un socle flexible de protection des services Web :

- La transmission de jeton de sécurité par des mécanismes standards. De multiples formats peuvent être utilisées : couple utilisateur/mot de passe, certificat X.509, ticket Kerberos, jetons XML : SAML.

- La signature numérique des messages pour assurer leur intégrité en vérifiant que le message SOAP n'a pas été altéré depuis qu'il a été signé en se fondant sur le standard *XML Signature.* Pour signer le message SOAP, l'émetteur a besoin de sa clé privée et pour que le récepteur puisse vérifier la signature du message SOAP, il a besoin de la clé publique de l'émetteur.

- Le chiffrement de ces messages afin de garantir la confidentialité des échanges en se fondant sur le standard *XML Encryption.* Le chiffrement du message SOAP crée un secret cryptographique qui est partagé avec le destinataire du message. Pour chiffrer le message SOAP, l'émetteur a besoin de la clé publique du destinataire. Et le récepteur a besoin de sa clé privée pour déchiffrer le message.

Chacune de ces fonctions peut être utilisée séparément ou s'associer pour adresser de nombreux scénarios d'architecture de services sécurisés. WS-Security ne définit pas les méthodes de chiffrement (*XML Encryption*) ou de signature (*XML Signature*), mais comment incorporer les informations spécifiques à ces standards dans l'en-tête d'un message SOAP. En outre, cette spécification permet de définir les mécanismes de transmission de multiples jetons de sécurité, à travers de multiples domaines de confiances, en utilisant de multiples formats de signature et de chiffrement.

Pour véhiculer les informations de sécurité, WS-Security définit donc un en-tête générique pour le message SOAP indépendant du mécanisme de sécurité mis en œuvre. Cet en-tête doit pouvoir héberger plusieurs jetons de sécurité permettant d'identifier l'appelant, des informations sur la manière dont le message a été signé ou chiffré et des informations pour localiser la clé, contenue ou pas dans le message.

Le schéma de cet en-tête est le suivant :

```
<xs:element name= "Security" >
        <xs:complexType>
                <xs:sequence>
                <xs:any processContents="lax"
                        minOccurs="0" maxOccurs="unbounded">
                </xs:any>
                </xs:sequence>
                <xs:anyAttribute processContents="lax"/>
        </xs:complexType>
</xs:element>
```

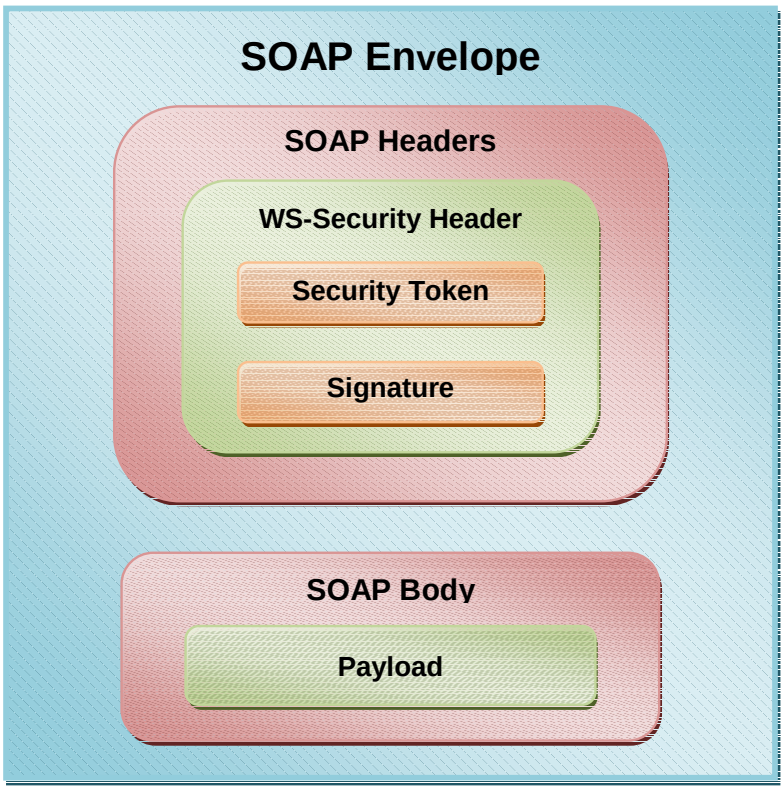


Figure 3.4 SOAP avec WS-Security

En outre, l'utilisation de WS-Security et des protocoles associés n'exclut pas l'utilisation d'un protocole de transport sécurisé, bien au contraire, il est tout à fait pertinent de cumuler les deux approches.

3.3.2.1. Transmission de jeton de sécurité

WS-Security permet de transmettre des jetons de sécurité via des systèmes hétérogènes par l'inclusion de jetons de sécurité fondée sur des mécanismes standards indépendants des mécanismes de transport spécifiques. La transmission du jeton de sécurité n'est pas nécessairement protégée de la lecture ou de la modification. Elle est fréquemment complétée par des mécanismes de signature et de chiffrement.

En se fondant sur la transmission de jetons de sécurité, WS-Security définit trois méthodes d'authentification extensibles à d'autres mécanismes :

1. Jeton basé sur le couple utilisateur/mot de passe
2. Jetons binaires : ticket Kerberos, certificat X.509
3. Jetons XML : SAML

Le tableau présente une comparaison de ces différents mécanismes :

Mécanismes	Avantages	Inconvénients
Utilisateur/mot de passe	Indépendant de l'environnement, de la technologie. Simple à mettre en œuvre.	Moins sécurisé.
Kerberos	Très sécurisé. Interopérabilité Kerberos.	Ne fonctionne qu'au sein d'un domaine Kerberos.
X.509	Très sécurisé. Interopérabilité.	Nécessite une PKI.
SAML	Mécanismes d'échange d'information d'authentification et d'autorisation. SingleSignOn sécurisé.	Peu d'implémentations existant aujourd'hui.

Tableau 3.1 comparaison des différents mécanismes

3.3.2.1.1. Jeton couple utilisateur/mot de passe

Bien qu'elle soit la technique qui offre le moins de garantie en termes de sécurité, cette approche est la plus utilisée. Elle est notamment mise en œuvre sur http dans le mode d'authentification « basic » ou « digest ».

Pour proposer ce modèle d'authentification le schéma WS-Security définit plusieurs types d'élément.

Le premier d'entre eux est l'élément UsernameToken[16] :

```
<xs:element name="UsernameToken">
    <xs:complexType>
        <xs:sequence>
            <xs:element ref="Username"/>
            <xs:element ref="Password" minOccurs="0"/>
        </xs:sequence>
        <xs:attribute name="Id" type="xs:ID"/>
        <xs:anyAttribute namespace="##other"/>
    </xs:complexType>
</xs:element>
```

3.3.2.1.2. Jeton binaire : Kerberos

Kerberos est un protocole standard d'authentification au sein du domaine, se basant sur une double authentification (aussi appelée authentification « mutuelle ») de l'identité de l'utilisateur et des services réseaux.

Son principe de fonctionnement est le suivant. L'utilisateur s'authentifie vis-à-vis du service KDC (*Key Distribution Center*). Le KDC délivre un *Ticket Granting Ticket* (TGT). Le TGT est un jeton qui doit, pour pouvoir accéder à d'autres ressources, être présenté au *Ticket Granting Service* (TGS) qui lui, délivre des tickets d'accès (*Session Tickets*) aux services réseau.

Ces tickets contiennent des données chiffrées incluant le mot de passe qui confirme l'identité de l'utilisateur vis-à-vis du service réseau sollicité. L'utilisateur peut alors utiliser la ressource conformément aux droits qui lui sont attribués. Le processus d'authentification est ainsi entièrement masqué à l'utilisateur.

[16] http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-profile-1.0.pdf

Ce mécanisme d'authentification peut être mis en oeuvre par les services Web. En effet, les messages SOAP peuvent passer directement un ticket Kerberos dans le header WS-Security. Le message SOAP véhicule le jeton Kerberos codé en base 64 dans un élément *BinarySecurityToken*[17] défini selon le schéma suivant :

```
<xs:element name="BinarySecurityToken">
        <xs:complexType>
                <xs:simpleContent>
                        <xs:extension base="xs:string">
                                <xs:attribute name="Id" type="xs:ID" />
                                <xs:attribute name="ValueType" type="xs:QName" />
                                <xs:attribute name="EncodingType" type="xs:QName"
                        />
                                <xs:anyAttribute namespace="##other"
                                processContents="strict" />
                        </xs:extension>
                </xs:simpleContent>
        </xs:complexType>
</xs:element>
```

3.3.2.1.3. Jeton XML : SAML

SAML définit un type de jeton de sécurité[18] dans le bloc d'en-tête <wsse:security>. Lorsqu'un récepteur traite un en-tête <wsse:Security> contenant ou référençant des assertions SAML, il sélectionne les signatures et les assertions SAML, selon une politique prédéfinie. La politique de sélection utilise les éléments <wse:SecurityTokenReference> présents dans les éléments <ds:KeyInfo> de la signature. Le receveur doit également établir une relation entre chaque sujet SAML et l'entité ("attesting entity") fournissant la preuve permettant de confirmer l'assertion. Les assertions SAML sont attachées aux messages SOAP par l'inclusion d'assertions ou de références à ces assertions dans l'en-tête <wse:Security> comme dans l'exemple suivant :

```
<xs:element name="BinarySecurityToken">
</SignedInfo>
<SignatureValue>WiMRRAbt79tNZtqhCDRnRdwPY1hVkwlPjXudg9x…
</SignatureValue>
<KeyInfo>
```

[17] http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0.pdf

[18] http://www.oasis-open.org/committees/download.php/16768/wss-v1.1-spec-os-SAMLTokenProfile.pdf

```
<wsse:SecurityTokenReference>
        <wsse:Reference URI="#SecurityToken-c9d59260-8721-4d35-8eff-
d47b5dc3e061" />
</wsse:SecurityTokenReference>
</KeyInfo>
</Signature>
```

SAML définit un élément *signature XML* pouvant contenir un certificat X.509 ainsi que la signature générée pour le contenu de l'élément. La signature peut être vérifiée en utilisant la clé publique contenue dans le certificat.

L'information de sécurité SAML est exprimée sous forme d'assertions à propos de sujets (un utilisateur ou un ordinateur) dotés d'une identité dans un domaine de sécurité. Ces assertions sont représentées sous forme de structures XML véhiculant des déclarations sur les opérations d'authentification déjà réalisées par ces sujets, et sur les autorisations d'accès de ces sujets à diverses ressources.

Ces assertions sont délivrées par des autorités SAML, selon un protocole définissant les formats XML de requête et de réponse, pouvant être lié à différents mécanismes de transport (comme SOAP).

3.3.2.2. Signature des messages

Signer un message permet de garantir que le message n'a pas été altéré au cours de l'échange et que l'émetteur du message en est bien l'auteur en se fondant sur l'identité véhiculée par le jeton de sécurité.

Avec WS-Security, l'intégrité du message se fonde sur l'utilisation conjointe de jetons de sécurité et du protocole ***XML Signature***[19] qui définit des mécanismes de signature chiffrée d'un message XML. En effet, la spécification *XML Signature* définit différents éléments dont l'élément <Signature> permettant de préciser le contenu d'une signature. Cette signature est le résultat d'un chiffrement du contenu du message SOAP et du jeton de sécurité, ce qui permet de valider cette signature, à la réception, par l'application d'un algorithme de déchiffrement.

WS-Security exploite cette spécification en ajoutant un élément <Security-TokenReference> à un élément <Signature> pour référencer le jeton de sécurité spécifié dans l'en tête <Security> du protocole SOAP. La spécification est conçue pour pouvoir facilement être étendue avec de nouveaux formats de signature. En outre, elle permet de gérer des signatures multiples sur le même échange.

[19] http://www.w3.org/TR/xmldsig-core/

3.3.2.3. Chiffrement des messages

Pour garantir la confidentialité des échanges, WS-Security offre des mécanismes de chiffrement et déchiffrement des messages qui se fondent sur l'utilisation conjointe de jetons de sécurité et du protocole ***XML Encryption***[20] pour chiffrer des parties du message SOAP.

Le chiffrement est la transformation mathématique des données d'un texte en clair en une version chiffrée illisible. La transformation requiert généralement des informations secrètes disponibles uniquement pour l'expéditeur et le destinataire. Ces informations portent le nom de « clé ». De cette manière, le message peut être chiffré par l'expéditeur et déchiffré uniquement par le destinataire à l'aide de la clé privée détenue par ce dernier.

Deux modèles de cryptographie sont proposés, le chiffrement symétrique et le chiffrement asymétrique :

- **Le chiffrement symétrique** : Une seule clé est utilisée pour chiffrer et déchiffrer le message. La clé partagée doit donc être transmise d'une manière sécurisée.

- **Le chiffrement asymétrique** : Utilise deux clés différentes pour le chiffrement et le déchiffrement. Aucun secret partagé n'est donc transmis sur le réseau.

WS-Security permet de chiffrer toutes composantes d'un message (corps, entête, pièces jointes) soit par le partage d'une clé symétrique entre émetteur et récepteur, soit par la transmission chiffrée de la clé dans le message.

[20] http://www.w3.org/TR/xmlenc-core/

3.3.2.4. Exemple

On peut par exemple prendre une application pour illustrer WS-Security. Une application exécutant l'addition de deux nombres va être appelée à partir d'un poste client, le service Web va exécuter l'addition et retourner le résultat au client.

Pour commencer, voila un échange TCP de l'appel sans implémentation du protocole WS-Security :

```
POST /WsApp/Service.asmx HTTP/1.1
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; MS Web Services Client Protocol
2.0.50727.312)
Content-Type: text/xml; charset=utf-8
SOAPAction: "http://tempuri.org/AddInt"
Host: adfsweb.treyresearch.net:3030
Content-Length: 307
Expect: 100-continue
Connection: Keep-Alive

HTTP/1.1 100 Continue

<?xml version="1.0" encoding="utf-8"?><soap:Envelope xmlns:soap="http://
schemas.xmlsoap.org/soap/envelope/" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema"><soap:Body><AddInt xmlns="http://
tempuri.org/"><a>1</a><b>2</b></AddInt></soap:Body></soap:Envelope>HTTP/1.1 200 OK
Date: Fri, 16 Nov 2007 08:42:24 GMT
Server: Microsoft-IIS/6.0
X-Powered-By: ASP.NET
X-AspNet-Version: 2.0.50727
Cache-Control: private, max-age=0
Content-Type: text/xml; charset=utf-8
Content-Length: 337

<?xml version="1.0" encoding="utf-8"?><soap:Envelope xmlns:soap="http://
schemas.xmlsoap.org/soap/envelope/" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema"><soap:Body><AddIntResponse
xmlns="http://tempuri.org/"><AddIntResult>3</AddIntResult></AddIntResponse></soap:Body></
soap:Envelope>
```

On voit donc les paramètres a et b qui sont envoyer à la méthode *AddInt* et le résultat qui est renvoyé par *AddIntResult*.

Maintenant en incluant WS-Security, celui-ci va donc rajouter un en-tête dans le message SOAP et le contenu du message sera chiffré et aussi signé si on le souhaite. Les données utiles du message sont inclues entre dans les balises <**CipherData**>.

Dans l'en-tête SOAP, on peut voir dans les balises <**xenc:EncryptionMethod**> l'algorithme de chiffrement qui sera utilisé, ici ce sera l'algorithme RSA et dans les balises <**ds:DigestMethod**> l'algorithme de hachage servant à la signature, ici l'algorithme utilisé est SHA1.

Voila une partie de la capture TCP effectuée :

```
POST /WsApp/Service.asmx HTTP/1.1
User-Agent: Microsoft WSE 3.0
Content-Type: text/xml; charset=utf-8
SOAPAction: "http://schemas.xmlsoap.org/ws/2005/02/trust/RST/SCT"
Host: adfsweb.treyresearch.net:2020
Content-Length: 10804
Expect: 100-continue
Connection: Keep-Alive

HTTP/1.1 100 Continue

<?xml version="1.0" encoding="utf-8"?><soap:Envelope xmlns:wsa="http://schemas.xmlsoap.org/
ws/2004/08/addressing" 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-200401-wss-
wssecurity-utility-1.0.xsd" xmlns:soap="http://schemas.xmlsoap.org/soap/
envelope/"><soap:Header><wsa:Action wsu:Id="Id-b19646d8-27b2-468c-b07e-382fe8136000">http://
schemas.xmlsoap.org/ws/2005/02/trust/RST/SCT</wsa:Action><wsa:MessageID
wsu:Id="Id-4b89fc90-71d3-47e9-aa64-afb2004c9d0d">urn:uuid:991506a8-aec4-4804-96f7-c7dcb214c7c0</
wsa:MessageID><wsa:ReplyTo wsu:Id="Id-2e5e3ab2-b53c-4bc2-b34d-ce7dfcbf2402"><wsa:Address>http://
schemas.xmlsoap.org/ws/2004/08/addressing/role/anonymous</wsa:Address></wsa:ReplyTo><wsa:To
wsu:Id="Id-ebfc9728-6513-467d-bb0e-79132904bea4">http://adfsweb.treyresearch.net:2020/WsApp/
Service.asmx</wsa:To><wsse:Security soap:mustUnderstand="1"><wsu:Timestamp
wsu:Id="Timestamp-9423c4bc-a431-4261-a064-43e67a1f8196"><wsu:Created>2007-11-16T08:46:29Z</
wsu:Created><wsu:Expires>2007-11-16T08:51:29Z</wsu:Expires></wsu:Timestamp><xenc:EncryptedKey
Id="SecurityToken-f7a2bb27-9156-4ff2-b341-69fcbc6fb9d3" xmlns:xenc="http://www.w3.org/2001/04/
xmlenc#"><xenc:EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-oaep-
mgf1p"><ds:DigestMethod xmlns:ds="http://www.w3.org/2000/09/xmldsig#" Algorithm="http://
www.w3.org/2000/09/xmldsig#sha1" /></xenc:EncryptionMethod><KeyInfo xmlns="http://www.w3.org/2000/09/
xmldsig#"><wsse:SecurityTokenReference><wsse:KeyIdentifier ValueType="http://docs.oasis-open.org/wss/
oasis-wss-soap-message-security-1.1#ThumbprintSHA1" EncodingType="http://docs.oasis-open.org/
wss/2004/01/oasis-200401-wss-soap-message-security-1.0#Base64Binary">xpr0F3AAVPZ850KxAa6L94W8/d8=</
wsse:KeyIdentifier></wsse:SecurityTokenReference></
KeyInfo><xenc:CipherData><xenc:CipherValue>bifq3rOHccTkboTqnas1qt0+TlMl1BKQODK5l1zSfpV2iJMlcRfppl9pP/
AXEVKliJOFWwkpzLc/2RD2pLRRBmXZpLub0LP8P7S/7N2TBdU8F4v3VFYrWFdEfkVQDHGKmfSZarukHG7A2oirsZoT3rqhnTndFZW4
ilBPFr48=</xenc:CipherValue></xenc:CipherData></xenc:EncryptedKey><wssc:DerivedKeyToken
wsu:Id="SecurityToken-dcded205-07d1-4c00-9042-c0bcdeed42ee" Algorithm="http://schemas.xmlsoap.org/
ws/2005/02/sc/dk/p_sha1" xmlns:wssc="http://schemas.xmlsoap.org/ws/2005/02/
sc"><wsse:SecurityTokenReference><wsse:Reference URI="#SecurityToken-f7a2bb27-9156-4ff2-
b341-69fcbc6fb9d3" ValueType="http://docs.oasis-open.org/wss/oasis-wss-soap-message-
security-1.1#EncryptedKey" /></wsse:SecurityTokenReference><wssc:Generation>0</
wssc:Generation><wssc:Length>32</wssc:Length><wssc:Label>WS-SecureConversationWS-SecureConversation</
wssc:Label><wssc:Nonce>NS3hf4Ynnre3mk7rp6/FZw==</wssc:Nonce></
wssc:DerivedKeyToken><xenc:ReferenceList xmlns:xenc="http://www.w3.org/2001/04/
xmlenc#"><xenc:DataReference URI="#Enc-ccc8f843-31d7-419b-bbe1-2d3ee8973b13" /><xenc:DataReference
URI="#Enc-4f674e0f-a1c6-4aa7-a4e4-780c06c04962" /><xenc:DataReference URI="#Enc-ca65e16e-f02a-488d-
b5a4-b127fe757ec9" /></xenc:ReferenceList><xenc:EncryptedData Id="Enc-4f674e0f-a1c6-4aa7-
a4e4-780c06c04962" Type="http://www.w3.org/2001/04/xmlenc#Element" xmlns:xenc="http://
www.w3.org/2001/04/xmlenc#"><xenc:EncryptionMethod Algorithm="http://www.w3.org/2001/04/
xmlenc#aes256-cbc" /><KeyInfo xmlns="http://www.w3.org/2000/09/
xmldsig#"><wsse:SecurityTokenReference><wsse:Reference URI="#SecurityToken-dcded205-07d1-4c00-9042-
c0bcdeed42ee" ValueType="http://schemas.xmlsoap.org/ws/2005/02/sc/dk" /></
wsse:SecurityTokenReference></
KeyInfo><xenc:CipherData><xenc:CipherValue>QkZAOy7H1aR6ZBzH0SgeQG8s3HOkTcDCm8567mGcqzYhc4cJTF5/
zpY0hEdAFEj+nJW846+iAnCnySVl08yca51tqWuApGb1QPzq8OHxxNsCCFxIxtcfZ5L8GNW8F+OJNl+4Vhj21GPkWS3F/
d7OCPiT80lPnJQCn/2EEZX5nHLVss7/Sg
+POfAajy6Jo3ZK4pVh0nnmM7fOtfqOEI8/7IdgzisHhDU9H6bMf76OouLAuPQZjgoZD4xcxz0
+DteyhYoXqBvUc8Wg7GR3OYubDj3VPxzcjRLiklebzNSsZb/
fdKCXTYz56xSyholKclicbRW87of42NcfRbOj2Wi5LWgsvC9rovEGLUnwOUU0gt4JNQTx+JBOV5v5a1KVdS0syof8QE9TOD/
NZOGt2ytUK+aPNduNEOHW374Kkw2qISCoG4SbRkX9A4ya+QhDQ2XwkzdXcB3/
hHKOoI7eYZjzDVkTBtOFzz0fPg1EGS8tSgt2d763mquXdOqAt6FT2x3j12uo44IW5S9dwRz7xoxb17eD/
IctWbRMRT41GG6yEO8u18eXKGzlhLUbgWDvVI80OF3NVeLbSaH+yId/
x49PhWuYFCopEgDtDlvbcdtEhxILtk9EKNqPi03R7u4sl9UJ2kAMg8mGwuCMAVFsqxOZmcLR8z5K2NSutK5zKboaQXD5FsRVy7GOSk
```

Tout le contenu utile du message est chiffré et donc ne peut pas être lu comme on le voit sur cette capture de l'échange TCP.

3.4. Bibliographie et Liens

- Christian Bernard, Libero Maesano,Xavier Le Galles. (2003).

  Services Web avec J2EE et .NET. Eyrolles.

  Chapitre 19 - Gestion de la sécurité

- Valérie Monfort, Stéphane Goudeau. (2004).

  Web Services et intéropérabilité des SI. Dunod

  Chapitre 5 - Sécurité des services Web

- **Open Web Application Security Project (OWASP)**

  http://www.owasp.org/index.php/Main_Page

- **SOAP Web Security**

  http://www.spidynamics.com/whitepapers/SOAP_Web_Security.pdf

- **Web Services Security Specifications Index Page :**
  http://msdn2.microsoft.com/en-us/library/ms951273.aspx

- **Understanding WS-Security :**
  http://msdn2.microsoft.com/en-us/library/ms977327.aspx

4. Active Directory Federation Services

ADFS[21] offre une solution de mise en œuvre de scénarios ***Web SSO*** et d'identité fédérée en se fondant sur les spécifications *WS-Trust* et *WS-Federation*. Avec cette technologie, chaque organisation authentifie ses propres utilisateurs et maintient ses comptes.

Lorsqu'un utilisateur souhaite accéder aux ressources d'une organisation partenaire, son organisation retransmet les preuves de son identité. Des droits lui sont octroyés en fonction du degré de confiance vis-à-vis de son organisation parente.

De cette manière, l'organisation partagent l'application n'a pas la gestion des comptes de ces partenaires, évitant ainsi les doublons, ainsi que l'augmentation du nombre de mots de passe.

Pour mettre en œuvre cette technologie, quatre scénarios vont être mis en place pour expliquer et montrer les différentes applications dans lesquelles une authentification ADFS peut être utilisée :

- Le premier sera un scénario dans lequel on accède à une application Web .NET. On accède aux informations sur l'utilisateur à l'aide de méthodes .NET qui nous permettent de lire le contenu du jeton ADFS.

- Le second scénario sera basé sur l'accès à une application Windows SharePoint Services pour les partenaires de l'organisation qui partage l'application. Une modification de l'application sera donc nécessaire pour activer l'authentification *WebSSO*.

- Le troisième scénario est basé sur l'accès d'un utilisateur à un Web Service en utilisant l'authentification ADFS.

- Le quatrième scénario aura pour but de montrer l'interopérabilité offerte par *WS-Federation* en remplaçant le serveur Web Microsoft par un serveur Web

[21] http://technet2.microsoft.com/windowsserver/en/library/050392bc-c8f5-48b3-b30e-bf310399ff5d1033.mspx?mfr=true

Linux muni d'Apache. L'accès à l'application sur le serveur doit donc se faire indépendamment du système d'exploitation de la machine.

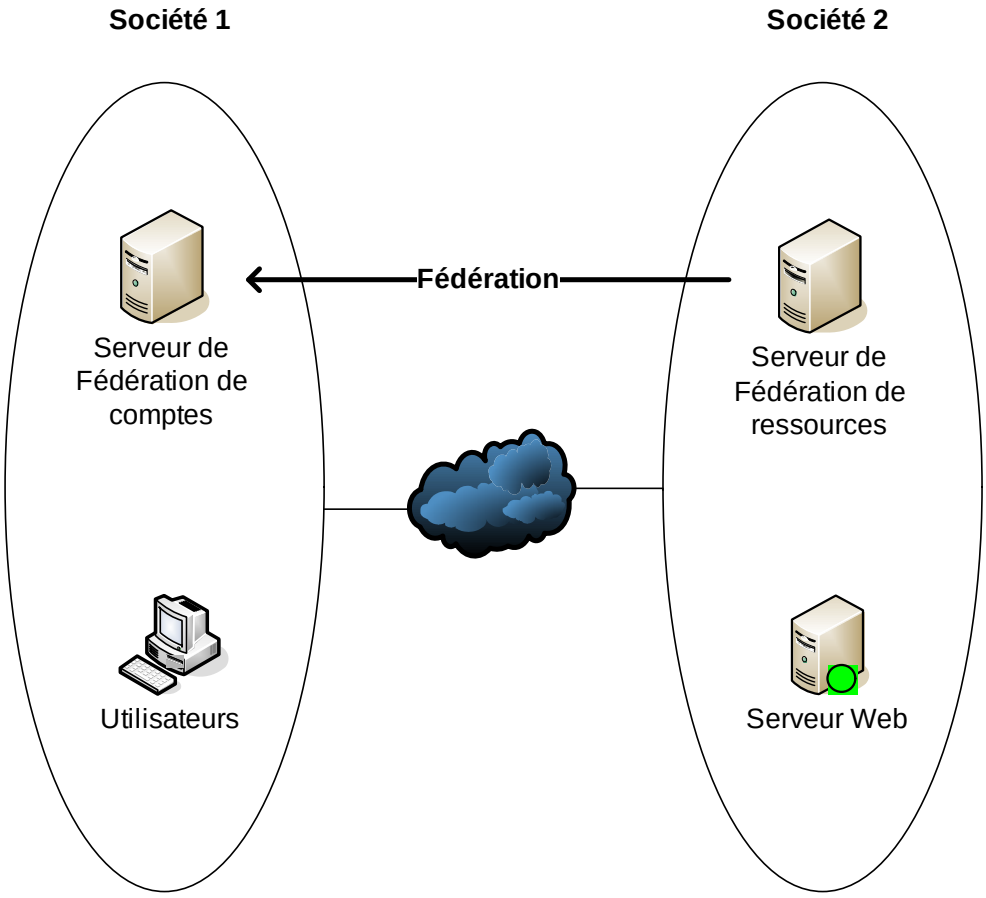


Figure 4.1 Architecture théorique ADFS

Cette architecture permet donc de partager en toute sécurité les informations d'identité d'un utilisateur au sein d'une organisation et entre les fédérations, sans créer ni maintenir des approbations externes ou des approbations de forêt entre ces organisations.

Dans une architecture réelle, on trouvera des firewalls pour sécuriser l'accès à l'entreprise par internet. Etant donné que tout le trafic ADFS est basé sur SSL et donc généralement sur le port 443 qui est le port par défaut SSL, il suffit d'une règle autorisant le trafic SSL à passer le firewall pour que le mécanisme ADFS puisse s'effectuer. Il n'y a pas d'autres ports à devoir ouvrir ce qui pourrait entrainer une diminution de la sécurité.

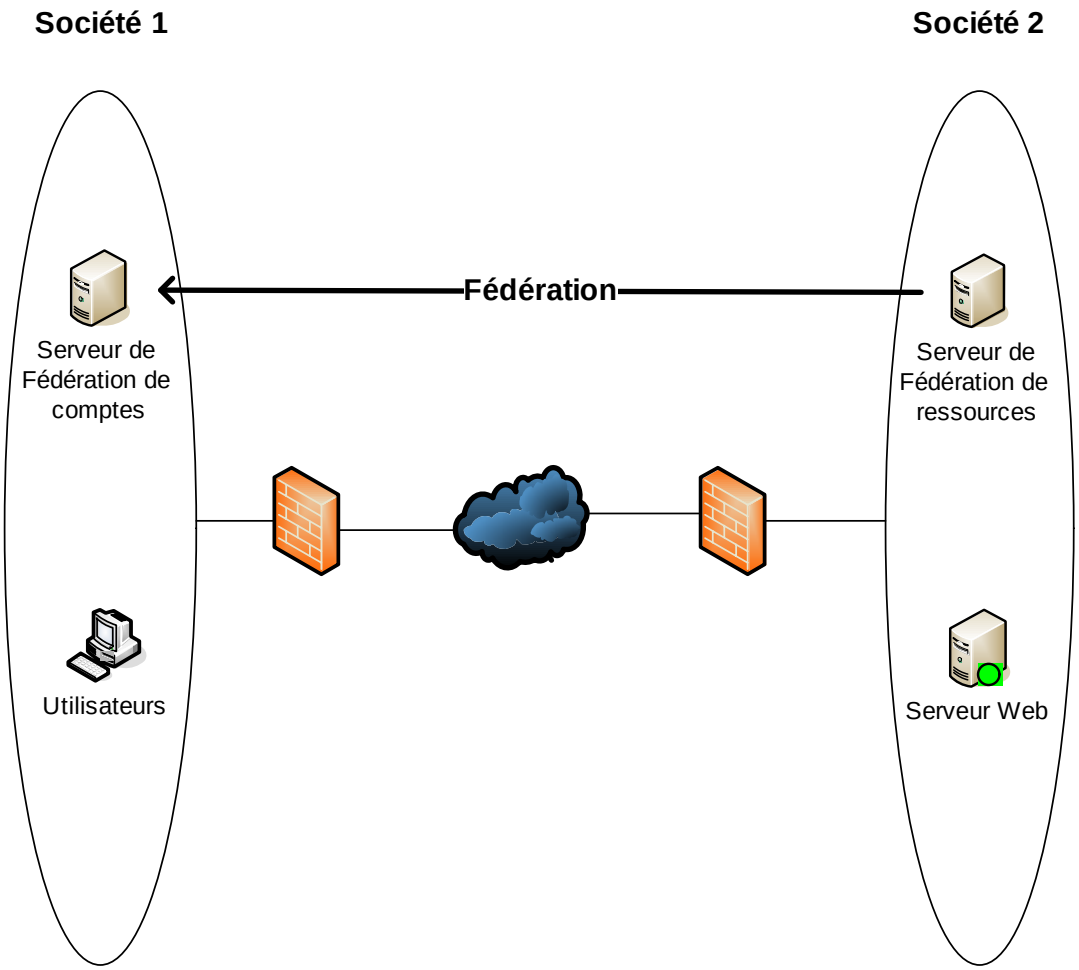


Figure 4.2 Architecture réelle ADFS

Un des avantages de l'architecture ADFS est qu'elle peut venir se superposer sur une architecture déjà présente dans une entreprise.

Une société qui comporte ses serveurs, ses contrôleurs de domaine, son annuaire AD, peut si elle le souhaite crée une fédération avec une société partenaire sans toucher à sa configuration actuelle.

Un serveur peut donc par exemple être rajouté dans la société qui fera office de serveur de fédération. Ce serveur de fédération va utiliser l'annuaire pour collecter les informations sur les utilisateurs qui se connecteront à l'application partenaire.

4.1. Architecture

L'architecture ADFS se compose de plusieurs acteurs[22] :

- **Utilisateurs** : Représente ceux qui se connecte à l'application.

- **Serveurs de fédération**[23] :

    - **Comptes** : Sert à l'ouverture de session des comptes d'utilisateurs locaux, dans un magasin Active Directory. Il est chargé d'émettre les jetons de sécurité contenant les différentes revendications de l'utilisateur.

    - **Ressources** : Valide les jetons de sécurité émis par le serveur de fédération de compte et en recrée un à l'intention du serveur Web.

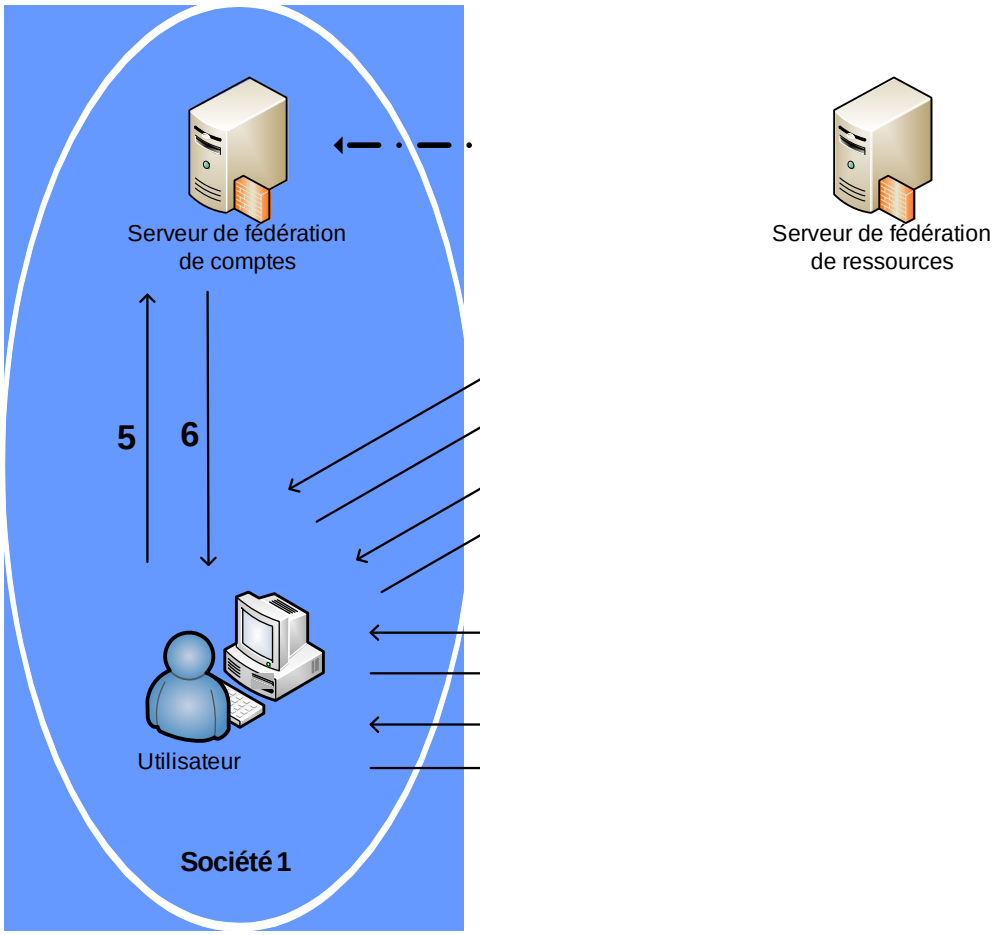
- **Serveur Web** : héberge le composant Agent Web ADFS[24] afin d'offrir un accès sécurisé aux applications Web qui sont hébergées sur le serveur Web. L'agent Web ADFS gère les jetons de sécurité et les cookies d'authentification qui sont envoyés à un serveur Web. Le serveur Web requiert une relation avec un service de fédération de façon à ce que tous les jetons d'authentification proviennent de ce service de fédération.

[22] http://technet2.microsoft.com/windowsserver/en/library/050392bc-c8f5-48b3-b30e-bf310399ff5d1033.mspx?mfr=true

[23] http://technet2.microsoft.com/windowsserver/en/library/050392bc-c8f5-48b3-b30e-bf310399ff5d1033.mspx?mfr=true

[24] http://technet2.microsoft.com/windowsserver/en/library/050392bc-c8f5-48b3-b30e-bf310399ff5d1033.mspx?mfr=true

4.2. Procédure de connexion



Lorsqu'un utilisateur de la société 1 se connecte à l'application de la société 2, le serveur Web regarde si l'utilisateur possède un cookie ADFS. Si c'est la première fois que l'utilisateur se connecte, le cookie n'est donc pas présent dans le navigateur de l'utilisateur. Le serveur Web redirige alors l'utilisateur sur le serveur de fédération de ressources.

Le serveur de fédération de ressources va alors demander à l'utilisateur d'indiquer de quel partenaire il fait parti pour le rediriger sur le bon serveur de fédération.
L'utilisateur choisit son appartenance et ensuite est donc redirigé sur son serveur de fédération de compte.

Le serveur de fédération de comptes va authentifier l'utilisateur soit en lui demandant d'indiquer son utilisateur/mot de passe ou soit en l'authentifiant directement en utilisant les mécanismes *Kerberos*. Le serveur de fédération va extraire les informations concernant l'utilisateur dans l'annuaire AD en fonction de l'accord de confiance établi avec le partenaire de ressources. Il va ensuite placer toutes ces informations dans un jeton SAML qu'il va finalement signer avec sa clé privée et rediriger l'utilisateur avec son jeton sur le serveur de fédération de ressources.

Le serveur de ressources vérifie que le jeton a bien été émis par le serveur de fédération de comptes de l'utilisateur grâce à la clé publique du serveur de fédération de comptes, il va en recréer un avec certaines modifications si besoin est, qu'il va signer avec sa clé privée et va rediriger l'utilisateur sur le serveur Web.

Le serveur Web va contrôler le jeton avec la clé publique du serveur de fédération de ressources et extraire les données de l'utilisateur contenues dans le jeton pour authentifier l'utilisateur et l'autoriser à exécuter l'application.

Tous ce processus est généralement transparent pour l'utilisateur. On peut faire en sorte que le serveur n'ait pas besoin de demander à l'utilisateur de quel partenaire il fait parti en indiquant l'adresse du serveur de fédération de comptes dans l'adresse lors de l'accès à l'application :

https://webserver/application/prog.aspx?whr=urn:federation:<*partenairecompte*>

où <*partenairecompte*> indique le domaine du partenaire de compte du client.

4.3. Cookies ADFS

Lors de la connexion au serveur, l'utilisateur est redirigé sur les différents serveurs de fédération pour obtenir le jeton avec les informations le concernant. Lors de ces diverses redirections les serveurs installent différents cookies[25] sur le navigateur de l'utilisateur pour offrir des fonctionnalités d'authentification unique pour éviter à l'utilisateur de rentrer ses informations à chaque fois qu'il se connecte à une application du partenaire de ressources.

Dans un environnement ADFS, il y'a 3 cookies :

- **Authentication/Token cookie (*_WebSsoAuth*)**
- **Home Realm Cookie (*_LSRealm*)**
- **Logout Cookie (*_LSCleanup*)**

4.3.1. Authentication/Token Cookie

Le cookie ***_WebSsoAuth*** contient les informations de connexion de l'utilisateur. C'est un cookie de session et donc il s'efface lorsque l'utilisateur ferme son navigateur.

Ce cookie contient le jeton SAML décerné par les serveurs de fédération. Le serveur Web contenant l'application va utiliser ce cookie pour extraire les informations de l'utilisateur.

Chaque cookie *_WebSsoAuth* est différent et donc l'application peut savoir lequel utilisé si il y'en a plusieurs étant donné qu'ils ont tous le même nom mais que le domaine ou le chemin du cookie est différent.

[25] http://technet2.microsoft.com/windowsserver/en/library/050392bc-c8f5-48b3-b30e-bf310399ff5d1033.mspx?mfr=true

4.3.2. Home Realm Cookie

Le cookie ***_LSRealm*** est obtenu après s'être correctement authentifie auprès du serveur de fédération du partenaire de ressource en lui indiquant de quel partenaire de comptes l'utilisateur fait parti. Cette adresse est sous la forme ***urn:federation:partenaire***. Lors de la première connexion le serveur demande à l'utilisateur d'indiquer de quel partenaire il fait parti. Lorsque l'utilisateur se reconnecte au serveur, il sera redirigé directement au bon endroit grâce au cookie dans lequel est indiquée la provenance de l'utilisateur.

4.3.3. Logout Cookie

Le cookie ***_LSCleanup*** est chargé d'aider à la déconnexion de l'utilisateur aux différents serveurs, il stocke les adresses des serveurs et applications visitées. Lorsque l'utilisateur se déconnecte, la page de *Logout* se charge d'aller sur les URL stockés et de déconnecter l'utilisateur de celle-ci. Le cookie ***_LSCleanup*** est aussi un cookie de session qui se supprime lorsque le navigateur est fermé.

4.4. Jeton ADFS

Le serveur de fédération de comptes aura donc la tâche de collecter les informations sur les utilisateurs dans l'annuaire AD et crée un jeton dans lequel il va intégrer ces informations.

Ces informations sont appelées "revendications" ou "*claims*". Elles vont donc être ajouter au jeton les une à la suite des autres.

La communication entre les différents acteurs de l'architecture se basant sur le protocole SSL, la communication est donc chiffrée. Etant donné que la communication est chiffrée le jeton n'est pas chiffré lui-même mais par contre il est signé avec la clé privée du serveur de fédération pour prévenir toutes modifications du jeton.

Ce jeton contenant les revendications est ensuite intégré dans un cookie qui est envoyé lors de la redirection de l'utilisateur.

Figure 4.3 Cookie ADFS

Chaque serveur de fédération possède une clé privée pour signer le jeton. Le serveur de fédération de comptes signe le jeton avec sa clé privée et le serveur de fédération de ressource vérifie la signature avec la clé publique du serveur de fédération de comptes.

Toute la phase d'authentification avec la méthode ADFS est donc basée sur le protocole SSL. La communication est donc chiffrée avec les certificats présents dans les serveurs de fédération. Ces certificats doivent généralement être demandés à une autorité de certification valide pour que la relation de confiance puisse être établie.

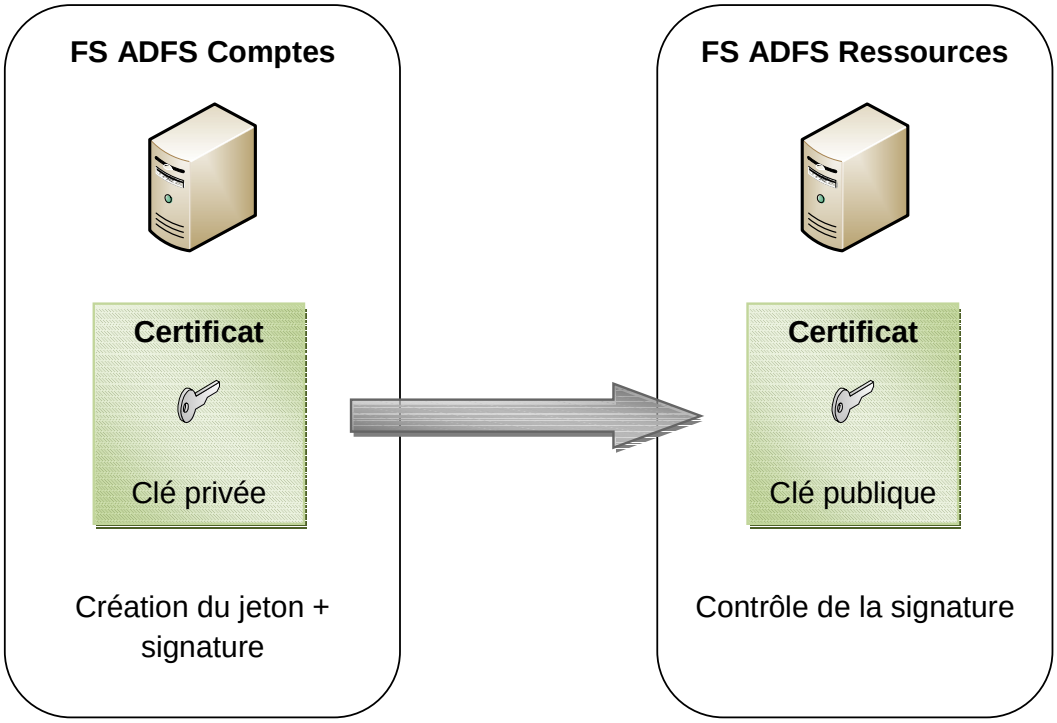


Figure 4.4 Procédure de signature et contrôle de la signature du jeton

4.4.1. Claims

Les *claims*[26] ou revendications sont donc les informations concernant les utilisateurs collectés dans l'annuaire AD.

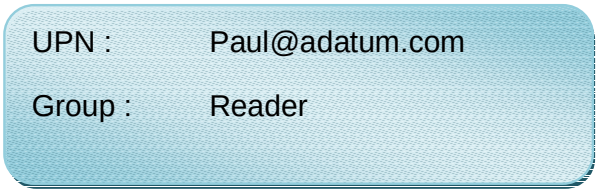
Dans l'application du partenaire de ressources, plusieurs actions peuvent par exemple être exécutées comme la consultation d'informations ou la possibilité de modifier certaines données.

Des privilèges spécifiques en fonction de la tâche à effectuer sont peut-être nécessaires. Un groupe peut être chargé de la commande de matériel et un autre de la lecture des informations. L'application doit donc pouvoir savoir quelles sont les privilèges qu'elle peut accorder à tel ou tel utilisateur. Elle ne peut pas utiliser le jeton Kerberos étant donné que le client est externe à l'entreprise et ne possède donc pas de ticket Kerberos.

[26] http://technet2.microsoft.com/windowsserver/en/library/050392bc-c8f5-48b3-b30e-bf310399ff5d1033.mspx?mfr=true

Pour résoudre ce problème, on va indiquer à l'application d'utiliser les informations contenues dans le jeton ADFS pour définir les droits accordés aux utilisateurs partenaires. Pour ça, il faut donc qu'il y'ait dans le jeton l'appartenance de l'utilisateur à un groupe spécifique.

Exemple d'un jeton d'administration



Exemple d'un jeton de lecture

Pour ajouter ces informations, on va configurer le serveur de fédération en lui indiquant les groupes présents dans l'annuaire AD qui vont devoir être inclus dans le jeton. Le serveur de fédération de comptes va ajouter le groupe dans le jeton, ensuite le serveur de fédération de ressources va extraire ces informations. Si le nom du groupe n'est pas le même entre les deux partenaires, par exemple, le groupe des lecteurs s'appelle "Reader" chez le partenaire de comptes et "Group_Reader" chez le partenaire de ressource, alors le serveur de fédération va mapper le nom à celui de son entreprise.

Finalement, le serveur Web va extraire ces informations et l'application va utiliser les revendications de groupes pour octroyer ou non les droits aux utilisateurs partenaires.

On peut créer des claims personnalisés pour indiquer des informations supplémentaires à l'application comme par exemple limiter le nombre de pièces que l'utilisateur chargée des commandes peut passer.

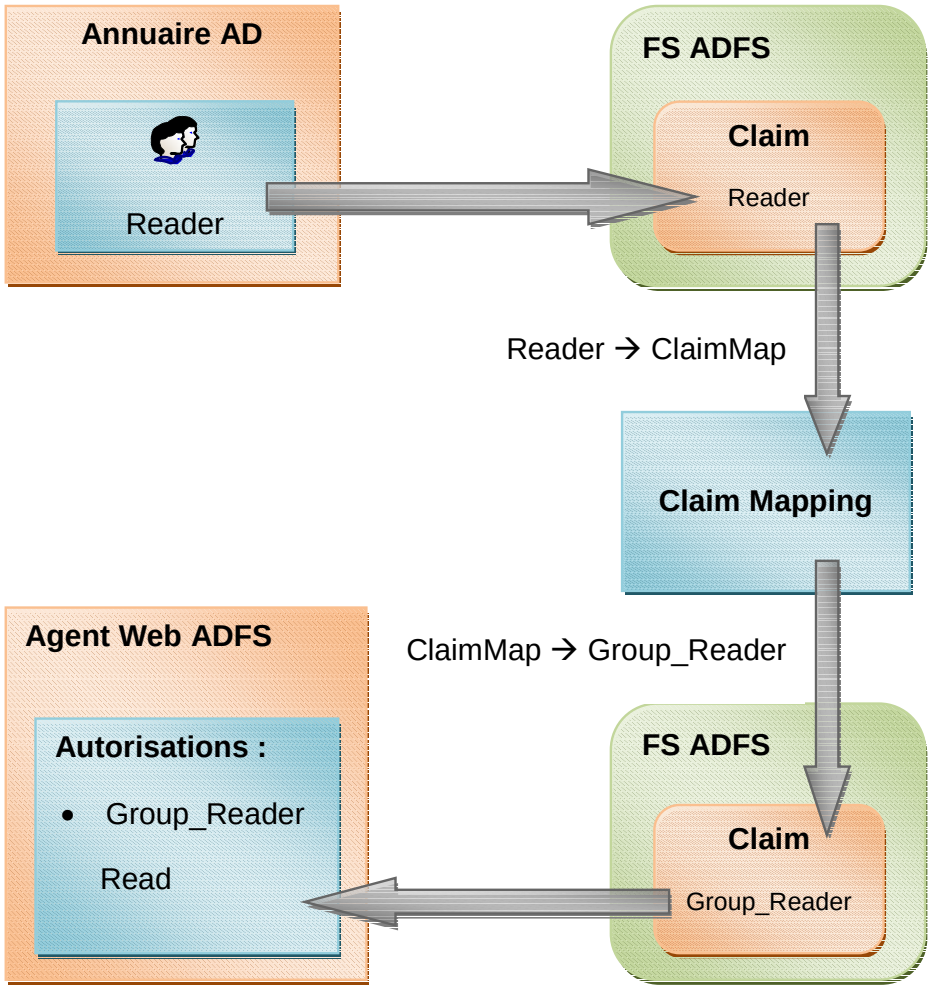


Figure 4.5 Exemple de la procédure pour l'utilisation des jetons ADFS

4.4.1.1. Claim Mapping

Lorsque le serveur de fédération du partenaire de comptes collecte les données de l'utilisateur dans l'annuaire AD, il va stocker ses informations dans un *Organization Claim.* Cet *Organization Claim* est un conteneur que l'on crée pour regrouper différents groupes que l'on veut associer à celui-ci.

On peut par exemple créer un *Organization Claim* qui s'appelle "***Readers***" et qui contient tous les groupes dans l'annuaire AD qui ont un accès en lecture à l'application partenaire.

Dans cet exemple le groupe AD "***Group_Readers***" est associé à l'*Organization Claim* "***Readers***" :

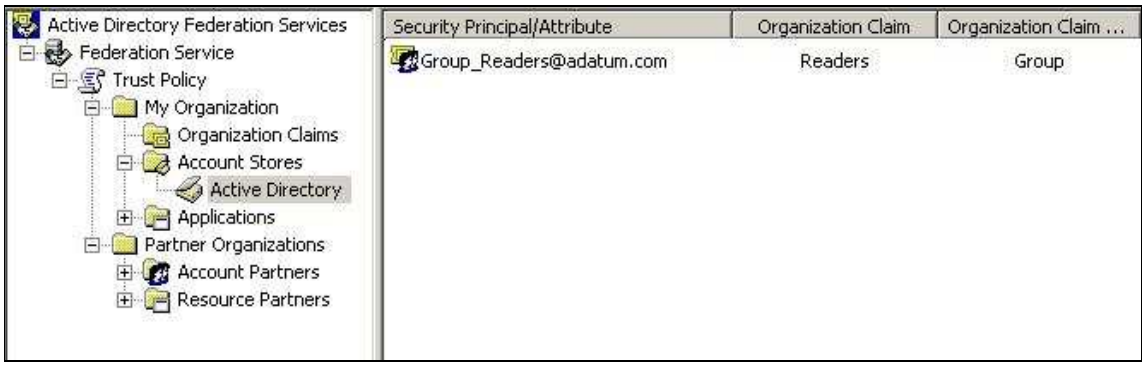


Figure 4.6 Groupe *Readers*

Ensuite on va associer (mapper)[27] cet *Organization Claim* à un "***Outgoing Claim***" qui est un jeton de sortie à destination du serveur de fédération du partenaire de ressources. Ici l'*Organization Claim "**Readers**"* est "mapper" à l'Outgoing Claim "**Readers_Map**" :

Figure 4.7 *Outgoing Mapping*

[27] http://technet2.microsoft.com/windowsserver/en/library/050392bc-c8f5-48b3-b30e-bf310399ff5d1033.mspx?mfr=true

Du coté du partenaire de ressources, le serveur de fédération va récupérer le jeton envoyé par le serveur de fédération du partenaire de comptes. On aura donc un "***Incoming Claim***", un jeton d'entrée, qui sera récupéré et ensuite mappé à un *Organization Claim* du partenaire de ressources. Le nom du jeton de sortie doit évidement être pareil que le nom du jeton d'entrée

Dans l'exemple, on va donc récupérer le jeton "***Readers_Map***" du partenaire de comptes que l'on va ensuite mapper à l'organization Claim "***Group_Readers***" que l'on aura au préalable crée :

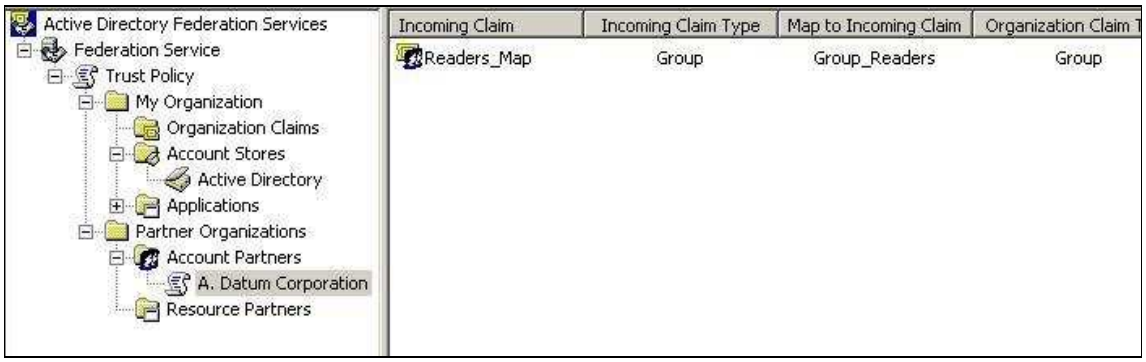


Figure 4.8 *Incoming Mapping*

Finalement l'application pourra venir lire les informations contenues dans le jeton "***Group_Readers***" qui contient donc les "Lecteurs" du partenaire de comptes.

4.5. Scénarios

4.5.1. Scénario 1

4.5.1.1. Présentation

Le premier scénario est l'accès à une application .NET distante portant l'extension ***.aspx*** mise à disposition par une organisation partenaire. Les utilisateurs se connectent à l'application et sont authentifiés directement par les différents mécanismes ADFS sans avoir à entrer un nouveau mot de passe pour avoir l'accès à l'application.

L'application utilise des instructions comme par exemple l'instruction .NET "***User.IsInRole("Role")***" pour savoir si l'utilisateur fait partis de tel ou tel groupe et donc de donner des autorisations à l'utilisateur dans l'application.

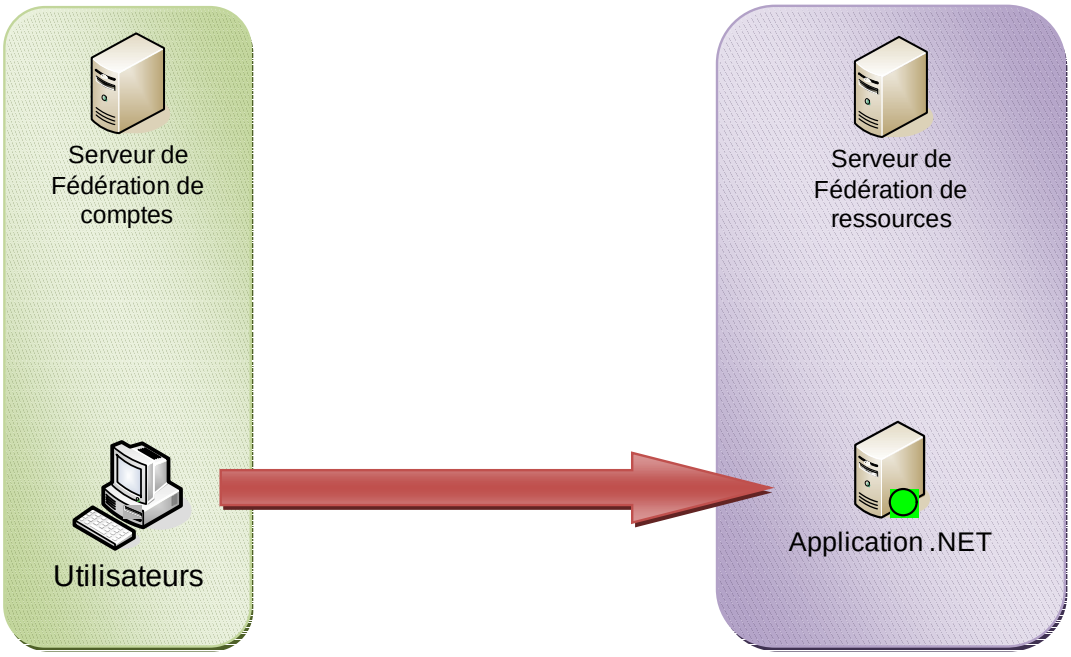


Figure 4.9 Architecture du scénario 1

4.5.1.2. Autorisations dans l'Application .NET

L'application va donc accueillir les utilisateurs de l'entreprise partenaire sans que ceux-ci aient un compte sur l'annuaire AD de la société qui partage l'application.

Pour connaître les différentes appartenances des utilisateurs à des groupes spécifiques, l'application va utiliser les jetons de sécurité ADFS. En fonction du groupe dans lequel l'utilisateur est placé, l'application va lui accorder ou non des autorisations.

Il y'a plusieurs types d'autorisations que l'on peut accorder à un utilisateur :

- **Lecture / Exécution de composantes de l'application**
- **Affichage / Accès à certaines parties de l'application**

4.5.1.2.1. Lecture / Execution

Dans le premier cas, on va limiter les actions que peux faire l'utilisateur en fonction du groupe dans lequel il se trouve.

Un utilisateur du groupe d'administration pourra par exemple modifier certains champs ou valeurs présentes dans l'application alors qu'un utilisateur du groupe des lecteurs ne pourra que lire ces valeurs mais pas les modifier.

Pour savoir dans quel groupe se trouve l'utilisateur, on va donc utiliser la méthode .NET ***User.IsInRole(Role)*** qui nous permet de savoir si l'utilisateur fait parti de tel ou tel groupe. Cette méthode s'utilise directement dans le code de l'application.

Exemple de l'utilisation de cette méthode :

```
If User.IsInRole("Admin")
        {
                // Vous faites parti du groupe Admin !
                // Code à exécuter pour les utilisateurs du groupe Admin
        }
```

Si l'utilisateur fait parti du groupe Admin, les instructions seront exécutées.

4.5.1.2.2. Affichage / Accès

Dans le deuxième cas, on va limiter l'accès à certaines pages ou parties de l'application à certains groupes d'utilisateurs.

On peut par exemple avoir un dossier privé contenant les pages de l'application ne pouvant être consultées que par les utilisateurs faisant partis du groupe des administrateurs.

Pour effectuer cette opération on va cette fois modifier le fichier ***web.config*** qui contient les paramètres de configuration de l'application. On va utiliser la balise ***<authorization>*** qui permet de définir les groupes autorises ou non a accéder à une certaine partie de l'application. On définit quels sont les parties de l'application que l'on veut restreindre avec la balise ***<location>***.

Exemple de l'utilisation de cette seconde méthode :

```
<location path="private">
        <system.web>
                <authorization>
                        <deny roles="Readers" />
                </authorization>
        </system.web>
</location>
```

Le dossier "private" à donc été interdit d'accès pour les utilisateurs du groupe "Readers". Tous les utilisateurs des autres groupes sont eux autorisés à accéder au dossier.

4.5.2. Scénario 2

4.5.2.1. Présentation

Le second scénario est l'intégration des mécanismes d'authentification ADFS pour l'accès à une application Windows SharePoint Services.

Par défaut on accède à l'application par une authentification Windows NT et donc par le biais de mécanismes *Kerberos*. Ces mécanismes ne sont pas présents lors de l'utilisation d'ADFS étant donné que l'on utilise à la place les jetons de sécurité qui contiennent les informations sur les utilisateurs distants.

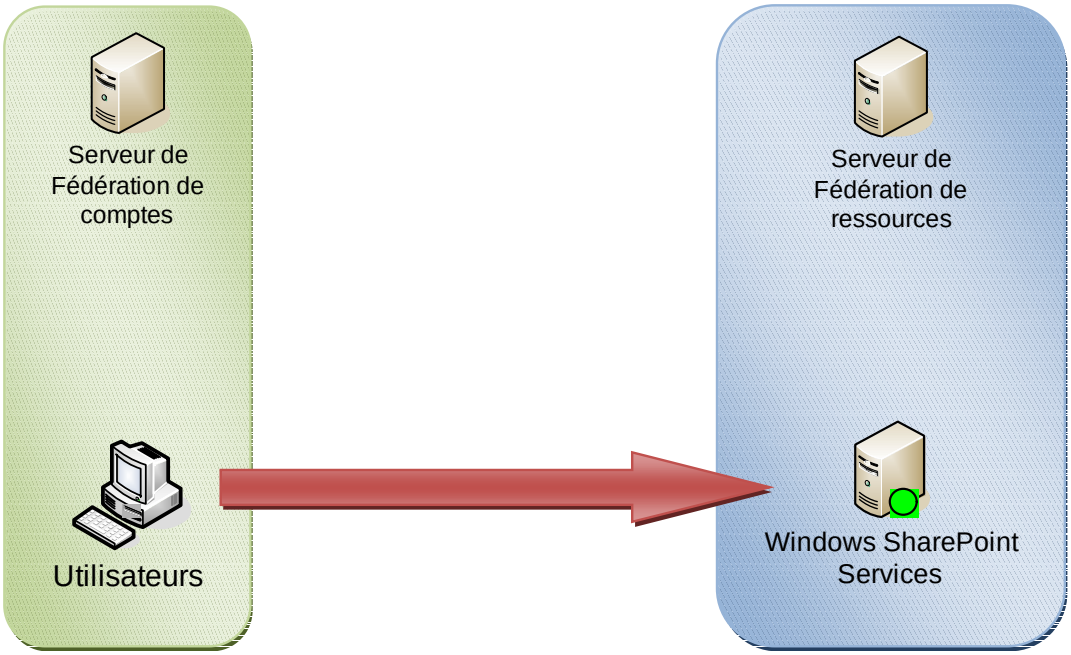


Figure 4.10 Architecture du scénario 2

4.5.2.2. Windows SharePoint Services

Windows SharePoint Services est un moteur de création de sites Web qui permet le partage des informations et le travail en équipe sur des documents. Ainsi, à partir d'un site WSS, une équipe va pouvoir gérer des informations telles qu'un planning, mais également de gérer les différentes versions d'un document, puisqu'un document possède un cycle de validation qui lui est propre, avant d'être publié.

WSS dispose des fonctionnalités suivantes :

- Création de pages grâce à des web parts développés en ASP.NET.
- Création de sites ou sous-sites liés à des équipes ou des projets.
- Système de contrôle de versions des documents stockés
- Moteur de recherche basique

Il se présente comme un site Web entièrement personnalisable :

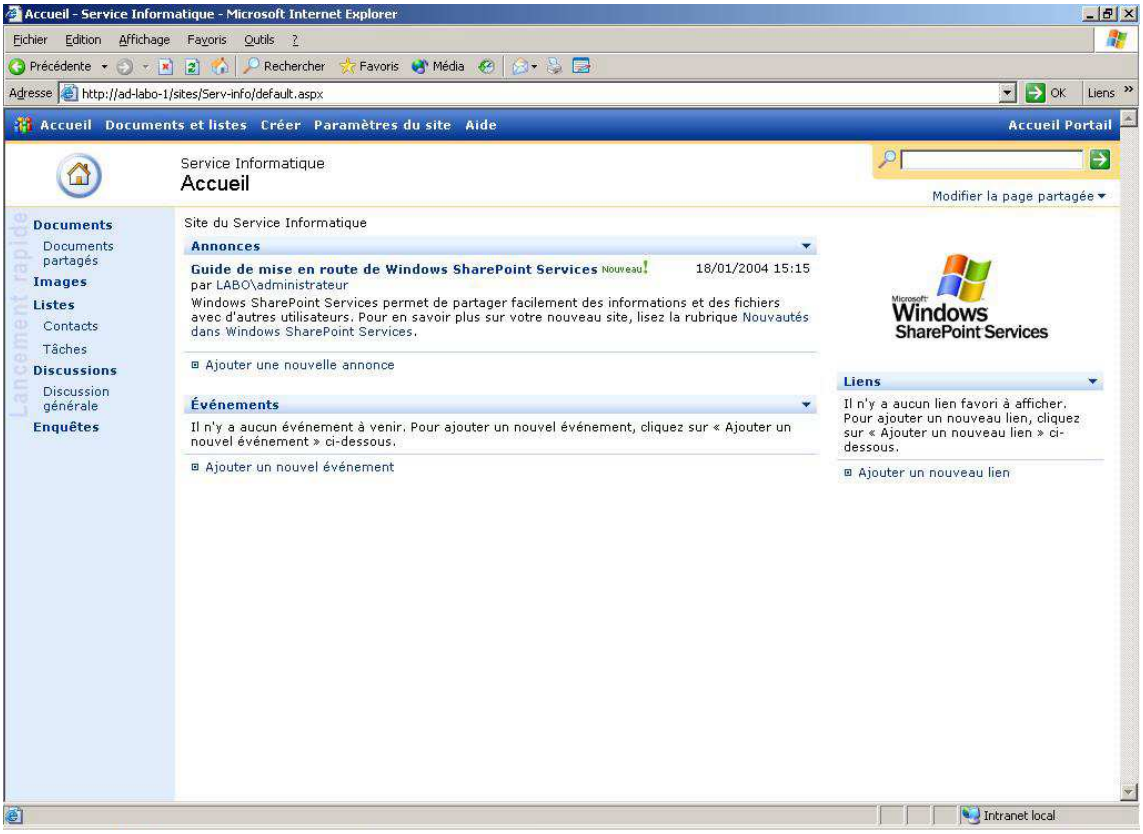


Figure 4.11 Windows SharePoint Services 2.0

Windows SharePoint Services utilise l'authentification Windows NT. Cela sous-entend donc qu'un contrôleur de domaine est présent dans l'entreprise avec un annuaire AD dans lequel sont stockées les informations sur les utilisateurs. Les utilisateurs sont authentifie auprès du contrôleur de domaine avec les mécanismes Kerberos et ainsi accèdent à l'application Windows SharePoint.

4.5.2.3. Intégration de l'authentification ADFS

Dans le cas d'une architecture ADFS, si les utilisateurs présents dans l'entreprise partenaire veulent pouvoir accéder à Windows SharePoint, ils doivent donc posséder un jeton Windows NT, ce qui sous-entend qu'ils possèdent un compte dans l'annuaire de l'entreprise qui héberge l'application. Dans ce cas là, tout l'intérêt de l'architecture ADFS tombe à l'eau.

Il va donc falloir modifier Windows SharePoint Services pour qu'il puisse lire les informations contenues dans les jetons ADFS et ainsi donner l'accès aux utilisateurs partenaires.

Pour effectuer cette opération on va utiliser Windows SharePoint Services 3.0 qui offre le support pour l'authentification Web SSO.

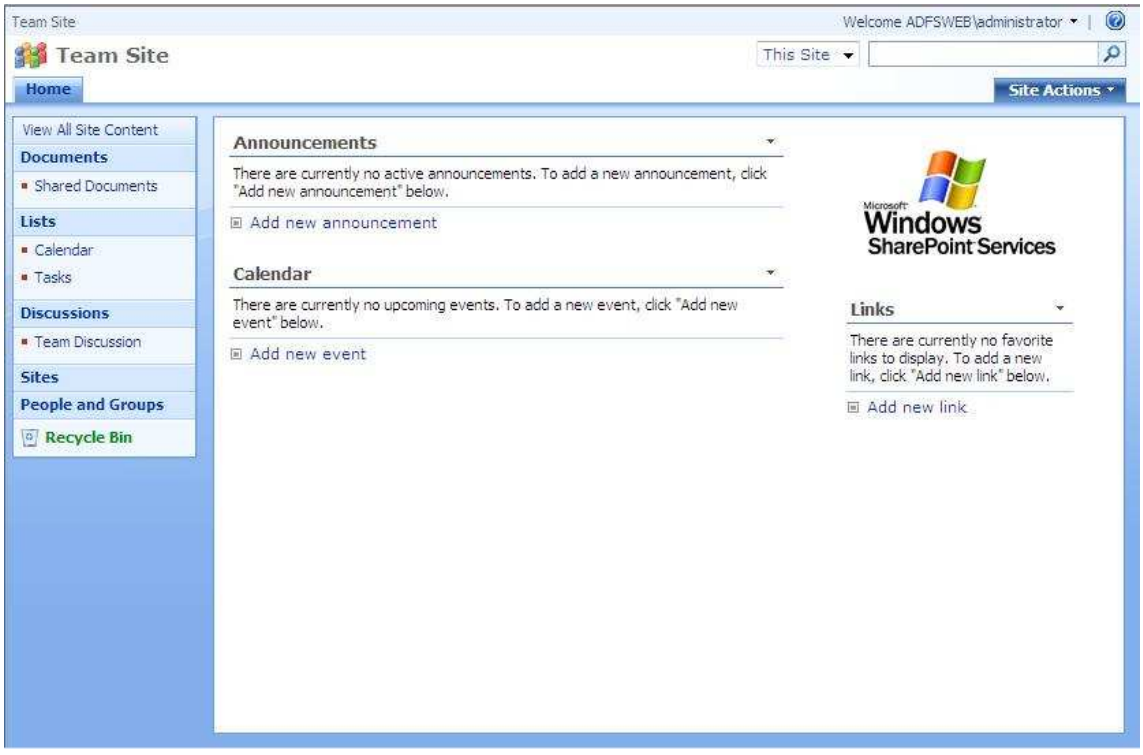


Figure 4.12 Windows SharePoint Services 3.0

Pour pouvoir utiliser les jetons ADFS dans Windows SharePoint Service, on va d'abord crée une nouvelle application Web. Cette application est accessible par l'intermédiaire d'un compte local pour le moment.

On va ensuite étendre l'application en créant une nouvelle zone qui elle, sera accessible grâce à une authentification WebSSO.

On aura donc 2 zones, une accessible via une méthode d'authentification traditionnelle Windows NT en utilisant les comptes stockés dans l'annuaire AD de l'entreprise et une, utilisant l'authentification WebSSO en accédant aux informations contenues dans les jetons ADFS.

Figure 4.13 Zones de l'application Web

On va donc configurer la zone accessible par les utilisateurs du partenaire de comptes pour qu'elle puisse accéder aux informations contenues dans les jetons, on lui indique d'abord qu'elle doit utiliser une authentification Web SSO et non une authentification du type Windows NT.

Figure 4.14 *Authentication Type*

Ensuite on va lui indiquer à quel endroit aller chercher les informations sur les différents groupes et utilisateurs à qui ont va attribuer les autorisations.

Figure 4.15 *Membership Provider Name* et *Role Manager Name*

On doit finalement modifier les différents fichiers *web.config* de l'application pour qu'on puisse accéder aux groupes et utilisateurs lors de l'attribution des autorisations.

Exemple d'un des ajouts à effectuer, qui indique où sont stockés les groupes :

```
<add name="SingleSignOnMembershipProvider2"
type="System.Web.Security.SingleSignOn.SingleSignOnMembershipProvider2,
System.Web.Security.SingleSignOn.PartialTrust, Version=1.0.0.0,
Culture=neutral, PublicKeyToken=31bf3856ad364e35"
fs="https://adfsresource.treyresearch.net/adfs/fs/federationserverservice.asmx
" />
```

On peut donc à ce moment accéder aux *Organization Claims* dans le serveur de fédération qui contiennent les groupes du partenaire de comptes qui ont l'accès à l'application partenaire. Des autorisations peuvent ensuite être attribuées aux différents groupes.

Figure 4.16 Permissions de l'application

4.5.3. Scénario 3

4.5.3.1. Présentation

Dans ce scénario, on va donc avoir un utilisateur qui va se connecter à un Web Service hébergé chez un partenaire de ressource. L'utilisateur se connecte à l'application Web qui porte cette fois l'extension ***.asmx***.

Pour authentifier l'utilisateur, les mécanismes d'authentification ADFS seront mis en œuvre. Le fichier *web.config* indique que l'on utilise une authentification ADFS et ensuite on va utiliser comme dans le scénario 1, la méthode "***User.IsInRole***" pour savoir si l'utilisateur est autorisé à utiliser le Web Service.

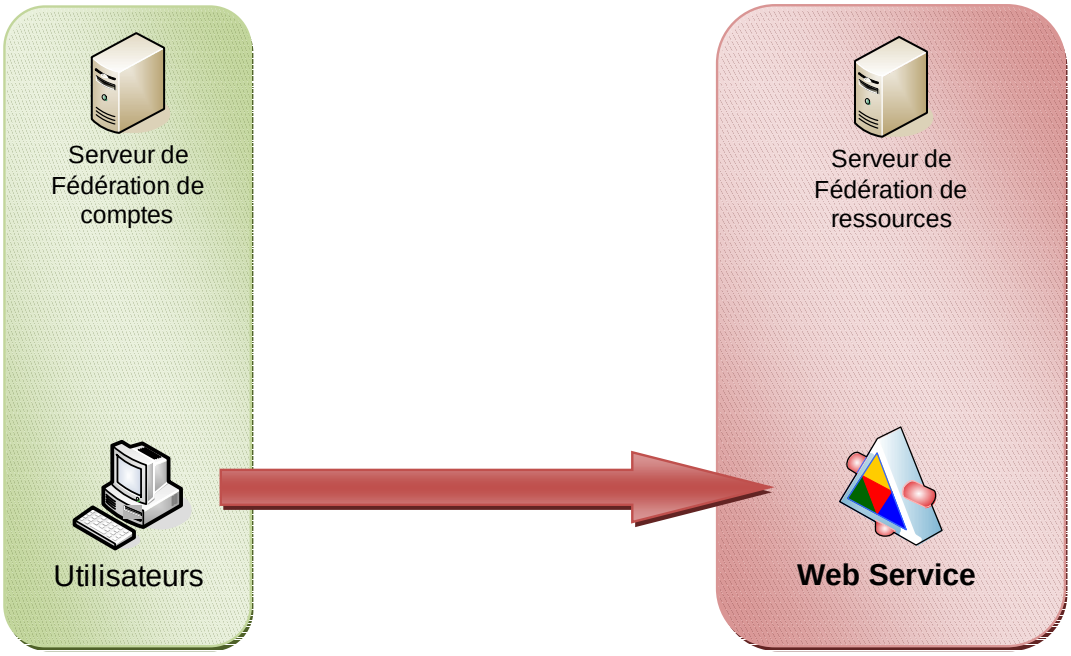


Figure 4.17 Architecture du scénario 3

W

L'application sera une simple addition entre 2 nombres que l'on rentre au clavier. Si l'utilisateur est autoriser à utiliser le service Web, le résultat de l'addition est affiché sinon un message disant que l'utilisateur n'est pas autoriser à utiliser l'application s'affiche.

Lorsqu'on se connecte à l'application, la page du Web service avec les différentes méthodes accessibles s'affiche. Ici Seul la méthode "**AddInt**" est présente qui est notre additionneur :

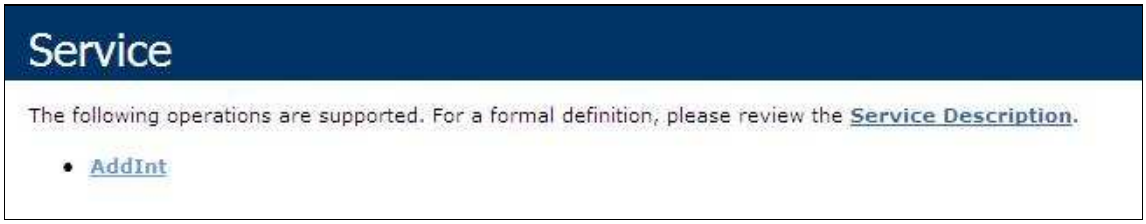


Figure 4.18 Page de démarrage du Web Service

Si on clique sur la méthode, on arrive sur la page qui nous permet d'utiliser la méthode :

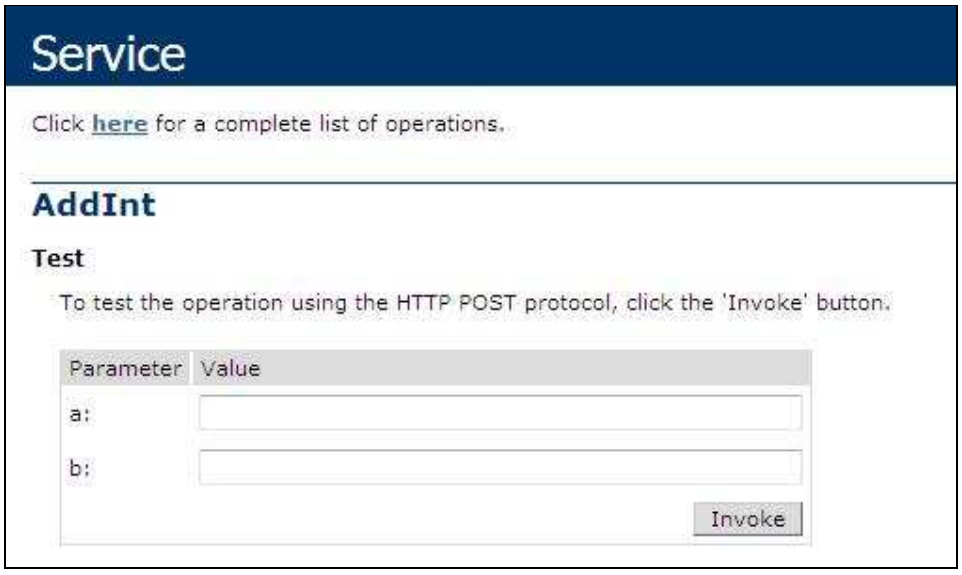


Figure 4.19 Page de la méthode

Les deux paramètres "a" et "b" sont les deux nombres que l'on doit renseigner pour effectuer l'addition.

A ce moment, si on est un utilisateur autorisé à utiliser le Web Service, lorsqu'on cliquer sur le bouton "*Invoke*", qui lance l'addition, une fenêtre s'affiche en indiquant le résultat.

Ici le résultat est 10 en ayant mis comme paramètre 5 pour a et b par exemple :

Figure 4.20 Résultat de l'addition pour un utilisateur autorisé

Si par contre l'utilisateur n'est pas autorisé à utiliser le Web Service, alors une fenêtre va s'ouvrir avec un message "*Access Denied*", que l'on aura configuré, indiquant à l'utilisateur qu'il n'est pas autorisé à utiliser le Web Service :

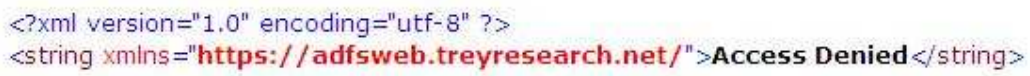


Figure 4.21 Résultat de l'addition pour un utilisateur non-autorisé

Si on regarde le code du Web Service, on remarque donc que l'on utilise la méthode "*User.IsInRole*" pour vérifier les informations qui sont contenues dans le jeton ADFS et vérifier que l'utilisateur fait parti d'un groupe autorisé à utiliser l'application.

```
[WebMethod]
public string AddInt(int a, int b) {

    int result = a + b;

    if (User.IsInRole("Gold"))
    {
            return result.ToString();
    }
    else
    {
             return("Access Denied");
    }

  }
```

Si l'utilisateur fait parti du group "Gold" le résultat est autorisé à s'afficher.

On peut créer par exemple une application Windows qui va nous permettre d'utiliser le service par l'intermédiaire d'une interface graphique. On entre les données dans l'application graphique et ensuite celle-ci va envoyer les informations au service Web qui va lui renvoyer la réponse et ainsi pouvoir l'afficher dans l'interface.

Voila un exemple d'une interface graphique pour l'addition de nos deux nombres :

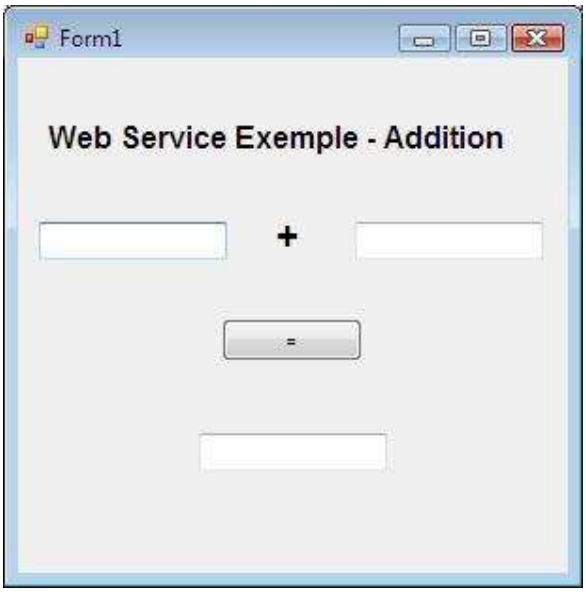


Figure 4.22 Interface graphique du Web Service

Cette interface ne peut pas être utilisée dans le cas d'une architecture ADFS. En effet, ADFS n'est à l'heure actuelle compatible qu'avec un navigateur web. L'interface graphique ne prend pas en charge les différentes redirections.

Si on essaie d'exécuter l'application avec ADFS, on obtient un message d'erreur :

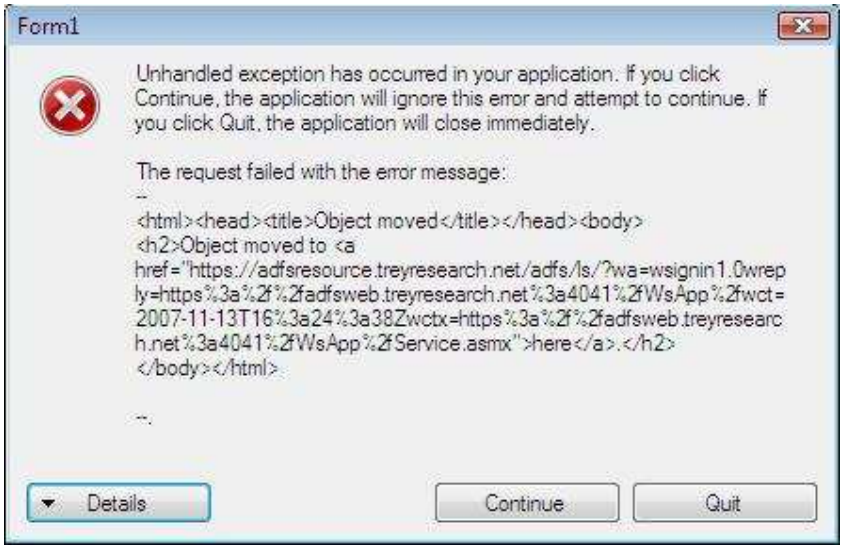


Figure 4.23 Message d'erreur suite à l'exécution de l'application

4.5.4. Scénario 4

4.5.4.1. Présentation

Le quatrième scénario permet de mettre en pratique l'interopérabilité offerte par *WS-Federation*. Dans les deux premiers scénarios, le serveur Web qui hébergeait l'application *.NET* et l'application *Windows SharePoint Services* était un serveur basé sur le système d'exploitation *Windows Server 2003*.

Dans ce troisième scénario, on va donc remplacer le serveur **Microsoft** par un serveur **Linux** avec un serveur Web **Apache**.

L'accès à l'application restera inchangé pour l'utilisateur qui ne verra pas de différence s'il a en face un serveur Microsoft ou Linux.

Les deux premiers sous-chapitres du chapitre "modules ADFS" sont juste une brève présentation de différentes sociétés qui mettent à disposition un module SSO ADFS pour Apache.

Le troisième chapitre présente le scénario qui a été réalisé pratiquement avec la solution qui a été choisie parmi les 3 présentées.

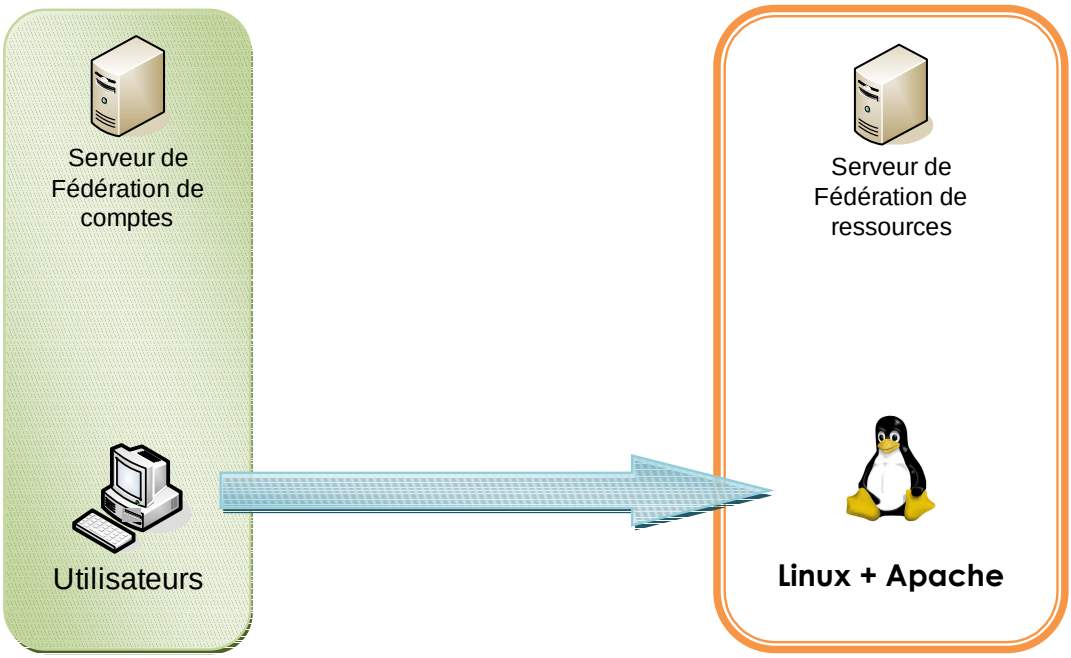


Figure 4.24 Architecture du scénario 4

4.5.4.2. Modules ADFS

Pour que le serveur Web Apache puisse comprendre les requêtes contenant les jetons de sécurité, on va devoir installer un module qui va donc remplacer l'agent Web ADFS Microsoft sur Windows Server 2003.

Plusieurs sociétés mettent à disposition un module à compiler que l'on installe donc sur le serveur Linux et qui va s'intégrer au serveur Web Apache :

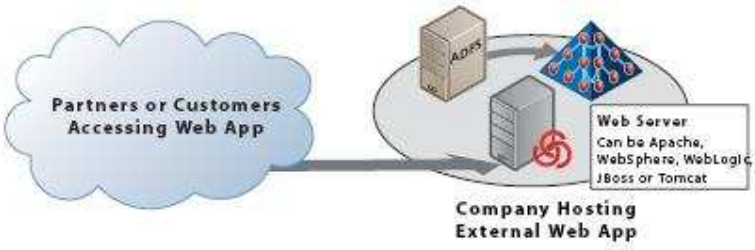
4.5.4.2.1. Centrify DirectControl

Centrify

http://www.centrify.com/

La société Centrify met à disposition un module ADFS "***DirectControl web SSO Agent***" sous la forme d'un paquetage RPM.

On installe l'agent sur le serveur Web et celui-ci va donc faire office d'agent Web pour rediriger l'utilisateur sur le serveur de fédération si besoin est.



4.5.4.2.2. Quest Software

http://www.quest.com/

Quest Software met à disposition un module "***Vintela Single Sign-on for Java***" qui va être intégré à Apache Tomcat pour les applications J2EE. Il se compose d'un certains nombres de fichiers prêts à l'emploi.

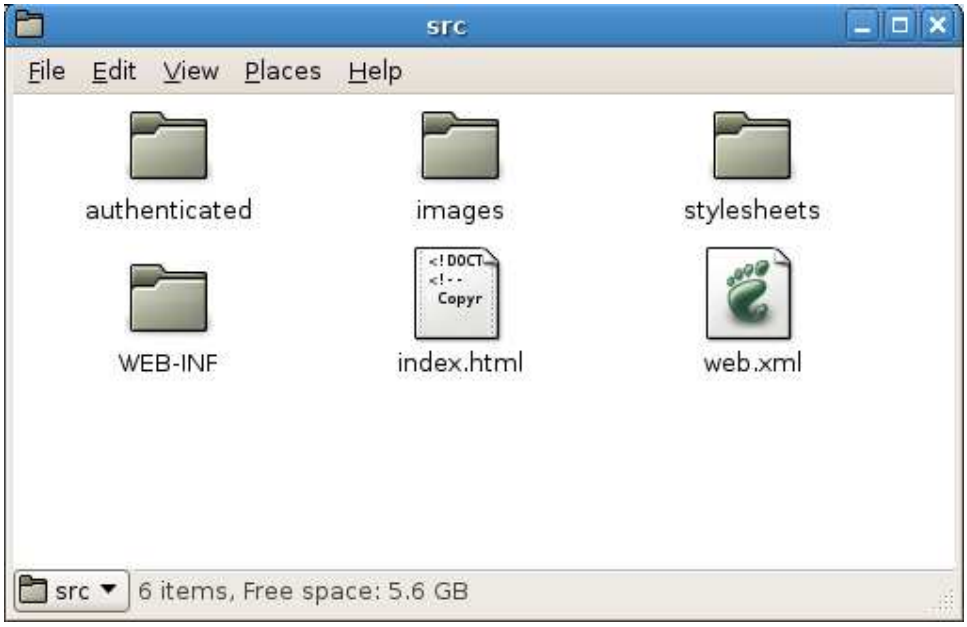


Figure 4.25 Fichiers de configuration et dossiers contenant les fichiers du module

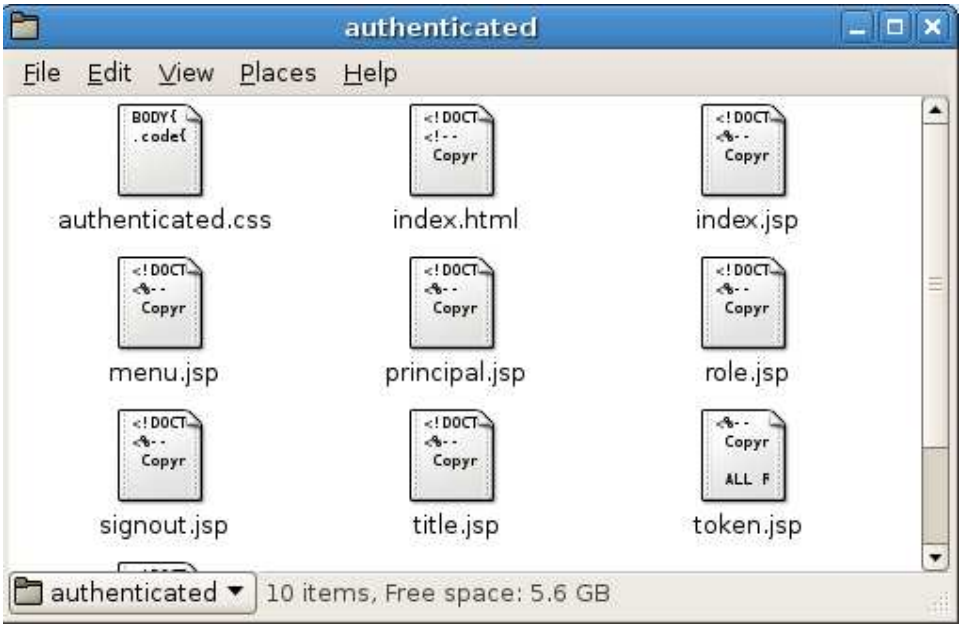


Figure 4.26 Fichiers du module

Après installation du module, lors de l'accès à l'application, une page nous indique que l'on est sur le serveur Web et nous propose de passer par une page html ou jsp pour l'authentification ADFS.

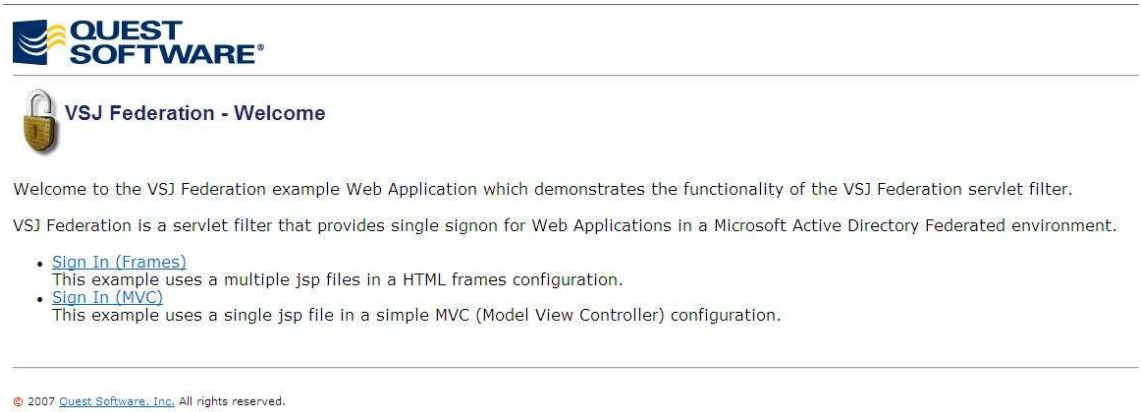


Figure 4.27 Page "index.html" du choix d'authentification

4.5.4.2.3. Ping Identity – Source ID

PingIdentity™ SourceID

http://www.pingidentity.com/ - http://www.sourceid.org/

Finalement, cette solution est celle qui à été mise en œuvre pratiquement.

Cette société met à disposition un module ADFS WsFedAuth "***WS-Federation for Apache 2.0 Toolkit***". Ce module est composé de plusieurs fichiers sources que l'on doit compiler manuellement pour créer le module que l'on va intégrer à Apache 2.0.

Figure 4.28 Fichiers sources

Après compilation de ces différents fichiers un fichier, on obtient un fichier **mod_auth_adfs.so** qui est une librairie dynamique équivalente à un fichier DLL sous Windows.

Figure 4.29 Librairie dynamique

Ensuite, on édite le fichier de configuration du serveur, pour lui indiquer diverses informations qui vont lui servir à déterminer quelles actions effectuer lors de la connexion de l'utilisateur :

```
SSLCertificateKeyFile /etc/apache2/ssl/ws0.key
SSLCertificateFile /etc/apache2/ssl/ws0.crt
```

Ces deux lignes indiquent l'emplacement des certificats du serveur.

```
WSFedAuthSTSURL https://adfsresource.treyresearch.net/adfs/ls/
```

Cette ligne indique l'adresse du serveur de fédération.

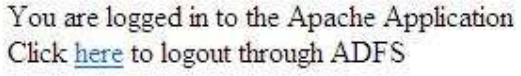
```
WSFedAuthCertFile /etc/apache2/ssl/resource_signing.cer
```

Cette ligne indique l'emplacement du certificat utilisé pour validé les jetons du FS

```
<Directory /protected-resources/>
        AuthType WSFedAuth
</Directory>
```

Cette suite d'instructions indique au serveur que lorsque un utilisateur accède au sous-répertoire "**protected-resources**", il doit effectuer une authentification en utilisant le module **WsFedAuth**, c'est à dire avec une authentification ADFS.

Finalement, lors de la connexion au serveur, si la configuration à été faite correctement, on accède à l'application hébergée sur le serveur Web Linux.



On peut afficher les informations contenues dans le jeton ADFS avec par exemple une instruction PHP "**$_ENV[AUTH_TOKEN_USER]**" qui sert à afficher le nom l'utilisateur authentifié :

Figure 4.30 Exemple d'extraction d'informations du jeton ADFS

4.6. WS-Federation

La spécification WS-Federation normalise la gestion des relations de confiance et de l'identité des utilisateurs ou des machines dans un même domaine authentifié pour créer une fédération de ces entités.

L'objectif est de proposer un mode de *Single Sign-On* (SSO) permettant à l'utilisateur d'être automatiquement identifié par les informations véhiculées dans le message.

La spécification WS-Federation se fonde sur les spécifications *WS-Security*, *WS-SecurityPolicy*, *WS-SecureConversation* et *WS-Trust* pour indiquer et identifier le type de niveau de confiance à relayer, en employant deux types de profils :

- **WS-Federation Active Requestor Profile** : gestion d'identité fédérée et authentification fondée sur les spécifications WS-Security.

- **WS-Federation Passive Profile** : modèle intégré pour fédérer l'identité l'authentification et l'autorisation à travers différents entités et protocoles de confiance.

Cette spécification définit ainsi les mécanismes permettant de gérer et relayer la relation de confiance pour fédérer la sécurité dans un environnement hétérogène.

Grâce à *WS-Federation* on peut donc faire interopérer plusieurs systèmes entre eux indépendamment des systèmes d'exploitation installé sur les machines.

4.7. Bibliographie et Liens

- Pellarin Anthony (2007)

  Web Service Federation – Active Directory Federation Services

  Travail de semestre

- Valérie Monfort, Stéphane Goudeau. (2004).

  Web Services et intéropérabilité des SI. Dunod

Chapitre 12.4 – Les futures spécifications de sécurité des services Web

- **Introduction à Active Directory Federation Services à destination des développeurs** : http://msdn.microsoft.com/msdnmag/issues/06/11/singlesignon/default.aspx?loc=fr

- **Vue d'ensemble des services ADFS :** http://technet2.microsoft.com/WindowsServer/fr/Library/8f4f4c4c-e80b-4101-b5fe-506339a265a61036.mspx?mfr=true

- **Configure Web SSO authentication by using ADFS (Windows SharePoint Services) :** http://technet2.microsoft.com/windowsserver/WSS/en/library/306b2d56-4419-432b-bf72-b4cae3845b001033.mspx?mfr=true

- **Step-by-Step Guide for AD FS in Windows Server 2008 :** http://technet2.microsoft.com/windowsserver2008/en/library/a018ccfe-acb2-41f9-9f0a-102b80a3398c1033.mspx?mfr=true

5. Conclusion

L'introduction des Web Services à donc changé l'approche des applications en effectuant le passage d'une architecture orienté objet à une architecture orienté service.

Aujourd'hui, on trouve un nombre conséquent de serveurs différents, de systèmes différents et le but serait donc de pouvoir faire communiquer touts ces machines entre elles. Grâce à des protocoles standardisés, les Web Services permettent une interopérabilité entre ces différents systèmes hétérogènes.

Mais les Web Services ont donc aussi apportés de nouvelles contraintes en matière de sécurité. Pour essayer de palier à ces contraintes, des protocoles de sécurité ont

été mis en place dont WS-Security permettant la signature et le chiffrement des messages.

La fédération entre entreprises et les scénarios B2B sont amenés à s'agrandir. Les sociétés auront de plus en plus à collaborer ensemble. Le partage d'application Web pour les collaborateurs ou les clients par exemple va être de plus en plus démocratisé.

Le nombre de mots de passe pour l'accès aux différentes applications devrait donc logiquement augmenté alors que cela va à l'encontre des règles de sécurité élémentaires.

Pour palier à cette problématique, l'architecture ADFS proposée par Microsoft permet donc d'étendre les informations AD sur les utilisateurs à Internet de manière transparente pour l'utilisateur et ainsi lui éviter de se réauthentifier et de devoir retenir un mot de passe supplémentaire pour chaque application. Cette architecture évite aussi à l'entreprise qui partage l'application de devoir héberger des informations sur les utilisateurs partenaires dans son propre annuaire.

Comme on l'a vu, ADFS v1 ne permet son utilisation qu'au travers d'un navigateur Web, on pourrait penser par exemple que dans ADFS v2, le support des applications Windows serait pris en considération.

Avec plus de temps à disposition, l'intégration des jetons ADFS dans les applications ainsi que l'accès aux données que le jeton contient aurait pu être poussée un peu plus loin avec l'utilisation par exemple de revendications personnalisées en plus des revendications "UID" et "group" pour pouvoir définir des valeurs personnalisées qui peuvent ensuite être utilisées dans les applications .NET par exemple.

Même si il reste encore des améliorations à effectuer au niveau de la sécurité notamment, les Web Services semblent être la solution appropriée pour résoudre les problèmes d'échange de données entre les systèmes hétérogènes et l'intégration d'applications. Pour les problèmes d'authentification Web SSO, ADFS semble donc être une alternative très intéressante qui offre des avantages tantôt à la réalisation qu'à l'exploitation.