



**Livre Blanc de sécurité SOA.
Sécurité des messages. v.1.0**

Project Documentation

Table of Contents

1 Sécurité Niveau Messages	
1.1 Introduction	1
1.1.1 Modèle sécurité simplifié SOA, J2EE, WS-*	1
1.1.2 WS-Security	5
1.1.3 WS-SecurityPolicy	7
1.1.4 WS-Trust	9
1.1.5 WS-SecureConversation	15
1.1.6 WS-Federation	17
1.1.7 SAML	19
1.1.8 XACML	20
1.1.9 XKMS	21
1.2 Best Practices	22
1.3 Les Solutions Open Source	23
2 Annexes	
2.1 Où trouver les spécifications?	24
2.2 Ressources de documentation	25

1.1 Introduction

«Les services Web XML vont rouvrir 70% des chemins d'attaques fermés par les pare-feu lors de la dernière décennie. Ils peuvent transporter virtuellement toutes les données utiles sur le port 80 et le pare-feu ne peut les arrêter.»

- Gartner Group , 2003

Les services Web apportent des bénéfices significatifs pour des applications basées sur l'Architecture Orientée Services, mais exposent des risques importants en terme de sécurité. Créer et gérer un environnement sécurisé pour les services web nécessite la manipulation et la maîtrise de spécifications et standards divers et variés ainsi que des technologies et logicielles et matérielles conséquentes.

Il convient d'étudier la mise en œuvre de la sécurité pour les Architectures Orientées Services (SOA) via les quatre volets suivants :

- **La sécurité niveau Transport** : pare-feu (firewall), VPN (Virtual Private Networks), authentification basique, non-répudiation et cryptage
- **La sécurité niveau Message** : Utilisation des jetons de sécurité afin de valider l'identité du consommateur du service ou du processus, utilisation des assertions d'autorisation pour valider l'accès au services.
- **Sécurité niveau application** : Sécuriser les composants appelés par les Web Services, EJBs, Servlets appelés via les services Web.
- **Sécurité niveau Données** : Cryptage et signature des messages afin de protéger les données stockées ou transmises.
- **Sécurité niveau Environnement** : Monitoring, logging et audit afin d'identifier les problèmes qui doivent être fixés et résolus et établir des communications sûres et fiables.

L'émergence des services web pose plusieurs problématiques dont celle de la sécurité des échanges de messages entre partenaires. Dans une architecture Orientée Services, les services web peuvent exposer des processus métier sensibles qui nécessitent un traitement particulier en terme de sécurité aux deux bouts du canal de communication. En plus, les services web sont des technologies récentes, ceci implique de nouvelles vulnérabilités et attaques ou menaces.

Les services web sont utilisés dans les cas suivants (la liste n'est certainement pas exhaustive) :

- Intégration de systèmes point-à-point
- Intégration d'applications entreprise
- Collaboration et partenariat Business
- E-Business
- Composition des processus métier
- Protection et ouverture des systèmes d'information
- Réduction des coûts du cycle de vie du développement et la maintenance des systèmes d'information

Les solutions de sécurité des services web doivent prendre en charge les concepts suivants :

- L'authentification
- L'autorisation
- La confidentialité
- L'intégrité
- La non-répudiation

En plus de ces concepts, le système de sécurité doit prendre en charge l'audit des actions et messages envoyés afin de tracer l'activité de la sécurité des web services.

Les utilisateurs des services web doivent être identifiés soit via un nom d'utilisateur combiné à un mot de passe soit via un certificat digital. Une fois l'utilisateur identifié, il doit posséder l'autorisation ou l'habilitation nécessaire afin d'effectuer le traitement qu'il a demandé. Toutes informations ou messages sensibles mis en jeu par le traitement doivent être confidentiels et ne doivent pas subir d'altération qui touche à son intégrité d'origine. Une fois le traitement exécuté, des mesures de non-répudiation doivent être mises en place afin d'éviter tout dénis des deux parts (consommateurs/fournisseur).

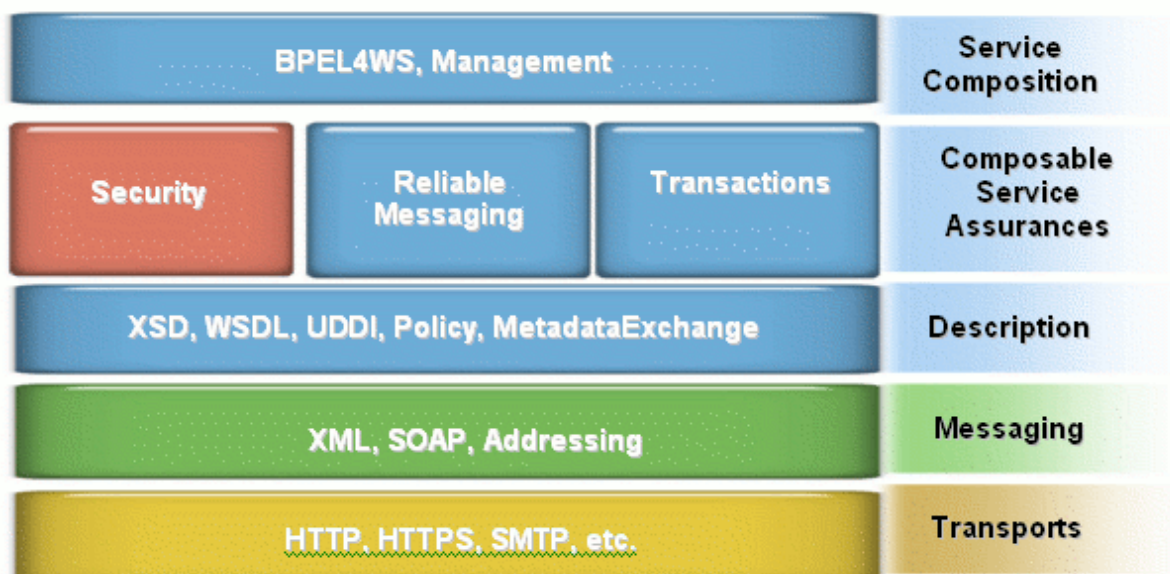
Les services web reposent sur des topologies applicatives diverses et variées telles que : l'Internet mobile, des passerelles, zones démilitarisées (DMZ), systèmes distribués La communication entre ces technologies s'effectue via des intermédiaires.

La sécurité n'était pas la priorité des organisations travaillant sur les spécifications de la stack WS. La sécurité au niveau de la couche de Transport n'étant pas suffisante, il demeurait un vide qui freinait l'adoption des services web. Heureusement, plusieurs propositions majeures ont été diffusées afin de combler ce vide dont **WS-Security** qui apporte un support globale de l'intégrité, de la confidentialité et l'authentification des messages et **SAML** qui définit un langage commun d'interopérabilité permettant de partager les informations liées à l'authentification et l'autorisation afin de faciliter la mise en place de fonctionnalités SSO et de permettre la délégation des droits. Il est intéressant à noter que d'autres propositions et spécifications tendent à émerger comme **Kerberos** , **XKMS** , **XACML** afin d'apporter un support complémentaire à la stack de sécurité.

La sécurité des messages est d'autant plus nécessaire si des intermédiaires sont présents dans une communication point-à-point. L'expéditeur d'origine et le destinataire final doivent établir des relations de confiance avec ces intermédiaires afin d'assurer la sécurité de bout en bout.

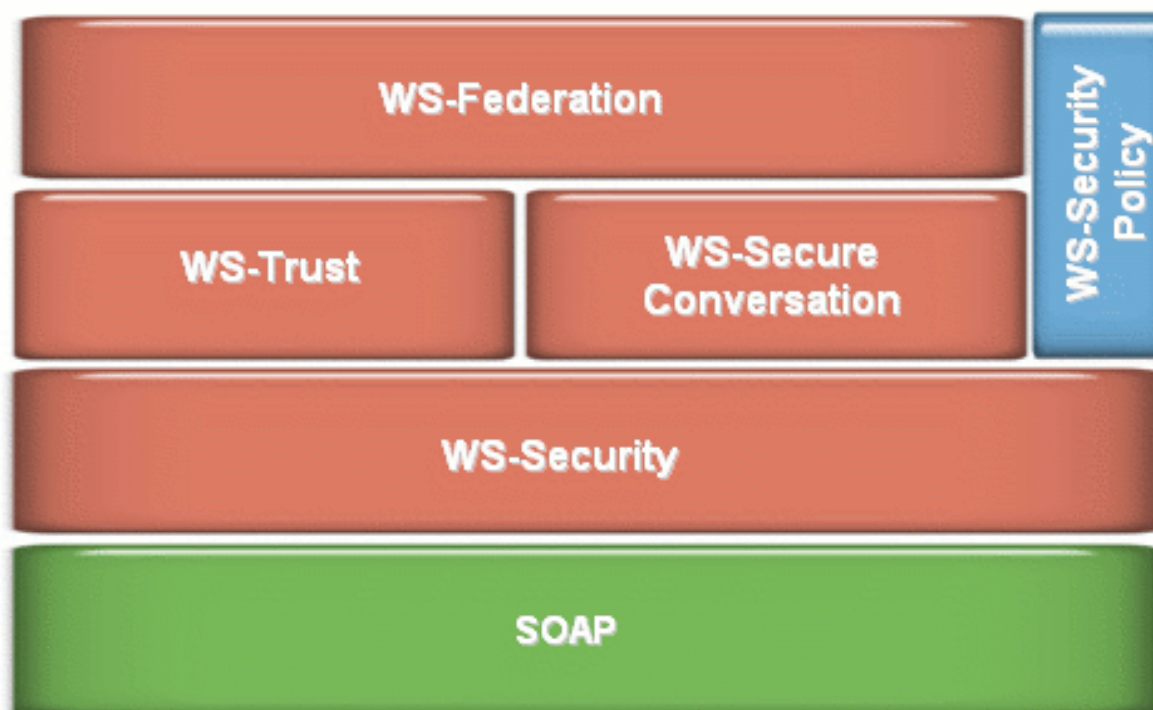
1.1.1 Modèle sécurité simplifié SOA, J2EE, WS-*

Dans ce chapitre on va décrire un modèle de sécurité faisant intervenir les concepts de l'architecture orientée services, des spécifications WS-* et plus particulièrement WS-Security et la plate-forme J2EE.



La sécurité des services Web fait partie intégrante de la stack des spécifications WS-*.

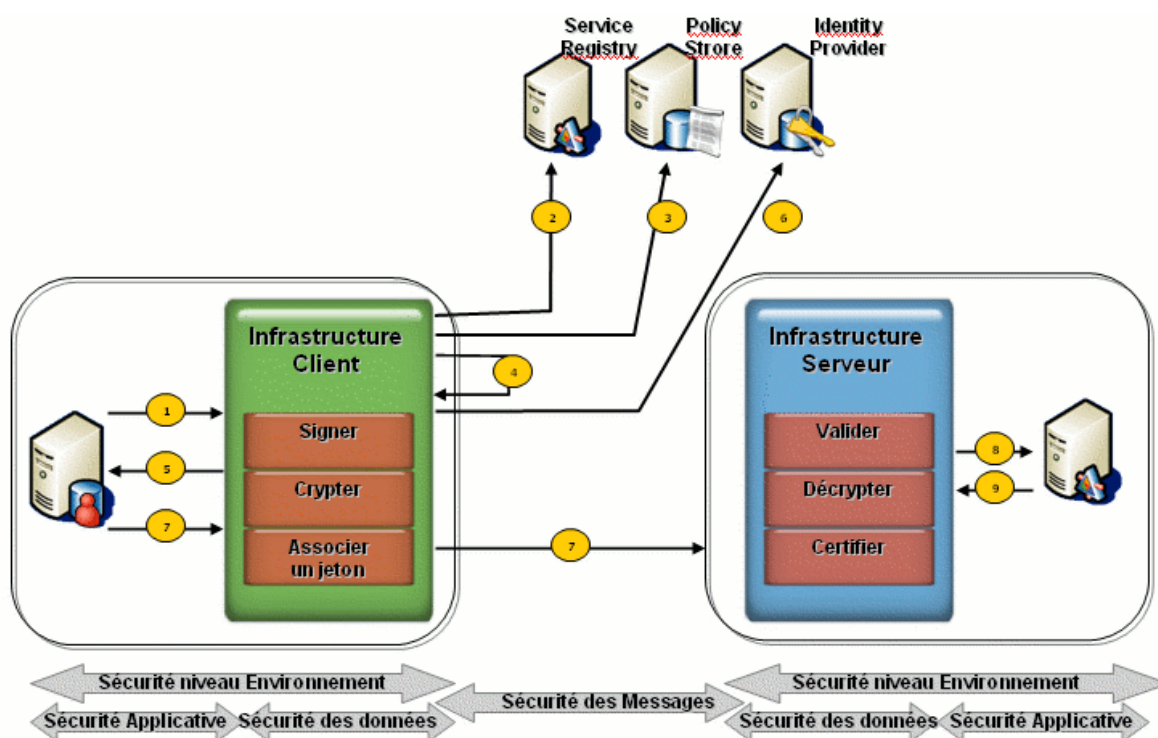
Plusieurs standards lui sont associés. Les plus importants et les plus stables en terme de spécifications sont cités ci-dessous. Pour plus de détails concernant chaque spécification se reporter aux autres livres blancs sur la sécurité niveau données et messages.



Afin de comprendre l'interaction entre les participants à une transaction basée sur un service Web, il conviendrait de dresser la cinématique relative au flux de messages transportés entre l'infrastructure Client et l'infrastructure Serveur. Ensuite seront associées les différentes spécifications, standards et outils associés à chaque étape de ce flux.

Les schémas mettent en jeu un client et un serveur, la cinématique peut être adaptée dans le cas d'un échange avec un intermédiaire ou plusieurs.

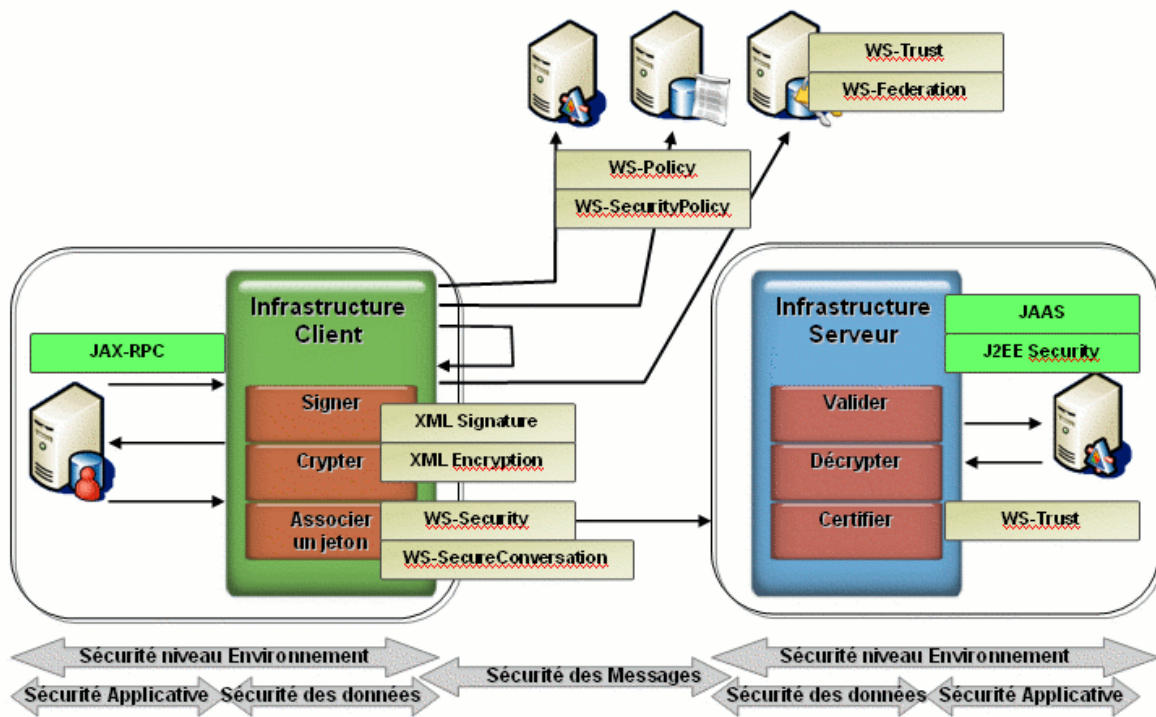
La cinématique ci-dessous ne fait pas référence au contexte de sécurité partagé et constitue un modèle dont la topologie reste simple et compréhensible.



Ci-dessous Le flux des messages

1. Le code client initialise l'environnement relatif à l'infrastructure client en récupérant des informations sur le proxy de connexion et soumet la requête désirée. Dans le contexte de la plate-forme J2EE, le client utiliserait l'api Jax-RPC.
2. Une fois l'infrastructure client alertée de la requête du client, cette dernière associe un processus à cette requête. Le processus correspond à un ensemble de services coordonnés et orchestrés. Pour chaque service, l'infrastructure client se connecte à l'annuaire des services et récupère la définition liée au service en téléchargeant le fichier WSDL représentatif.
3. une fois le fichier WSDL récupéré, l'infrastructure client se connecte à l'annuaire de policy (politique ou stratégies) puis récupérer les stratégies associées à l'appel du service Web. Ces stratégies peuvent contenir des exigences en terme de sécurité.
4. L'infrastructure Client Interprète les stratégies de sécurité de l'invocation du service, par exemple savoir quelle partie du message doit être cryptée. En ayant le fichier de policy, l'infrastructure client identifie le type de jetons de sécurité attendus par l'infrastructure serveur.
5. L'infrastructure client demande au client de spécifier son identité afin de l'autoriser.
6. Une fois l'identité du client validée, un jeton de sécurité est récupère via le fournisseur d'identité.
7. Lors de cette étape, le client constitue le message de la requête SOAP. Il signe le message, crypte les parties sensibles et associe le jeton de sécurité à l'entête du message. Une fois le message sécurisé créé il est enfin envoyé à l'infrastructure du serveur.
8. L'infrastructure serveur s'occupe de valider le message de le décrypter et de vérifier le jeton de sécurité afin d'installer une relation de confiance. Une fois la confiance établie, le service final est invoqué.

Ci-dessous sont associés à chaque étape de cette des technologies, standards et spécifications.



1.1.2 WS-Security

Les spécifications WS-Security permettent de définir de nouvelles extensions au protocole SOAP (entêtes de messages) qui fournissent un QoP (Quality Of Protection) via l'intégrité et la confidentialité des messages. WS-Security permet aussi d'attacher et d'inclure des jetons de sécurité (security tokens) dans un message SOAP. Elles comportent aussi un mécanisme pour spécifier des jetons de sécurité encodés et binaires. Ces mécanismes peuvent être utilisés indépendamment ou en combinaison afin de s'accommoder aux multiples modèles de sécurité et technologies de cryptage.

WS-Security est conçu pour être extensible, c'est à dire capable de supporter plusieurs :

- types ou formats de jetons de sécurité,
- domaines de confiance,
- formats de signature
- technologies de cryptage
- modèles de sécurité

Il permet aussi de prendre en charge des jetons de sécurité tels que :

- Nom utilisateur/Mot de Passe
- Certificat X.509
- Ticket Kerberos
- Assertion SAML

Les spécifications WS-Security protègent contre :

- L'altération des messages, en transmettant des signatures digitales pour toutes les parties du corps de SOAP (Body) et son entête (header)
- La divulgation des messages, en supportant le cryptage des messages.

Elles aussi être utilisées afin :

- De préserver l'intégrité des messages en utilisant des algorithmes de création de clefs,
- D'authentifier les messages via l'utilisation de multiples types de jetons de sécurité.

L'exemple ci-dessous illustre l'utilisation de WS-Security au sein d'une entete SOAP. Le namespace s11 est relatif à la vesion 1.1 de SOAP, pour utiliser la version 1.2 de SOAP, il convient d'utiliser

```
xmlns:s12="http://www.w3.org/2003/05/soap-envelope"
```

```
<?xml version="1.0" encoding="utf-8"?>
<s11:Envelope xmlns:s11="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:wsse="http://www.docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd">
<s11:Header>
  <wsse:Security>
    ...
  </wsse:Security>
</s11:Header>
<s11:Body>
  ...
</s11:Body>
</s11:Envelope>
```

Un jeton de sécurité correspond à un certificat qui prouve l'identité du consommateur de service web. WS-Security ne définit pas de stratégie d'authentification mais la manière dont sont transmis ces certificats.

Il existe plusieurs manières de transmettre un certificat :

- Le premier consiste ne l'utilisation de l'élément UsernameToken qui fait référence à un nom d'utilisateur accompagné d'un mot de passe qui pourrait être utilisé au sein d'une connexion de type SSL.
- La seconde consiste à envoyer un jeton de sécurité binaire. L'exemple ci-dessous utilise un ticket Kerberos ; il est renseigné dans l'élément <BinarySecurityToken>

```
<wsse:Security
xmlns:wsse="http://www.docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd">
<wsse:BinarySecurityToken
ValueType="http://www.docs.oasis-open.org/wss/2004/07/oasis-000000-wss-kerberos-token-profile-1.0#Ker
EncodingType="http://www.docs.oasis-open.org/wss/2004/01/#oasis-200401-wss-wssecurity-secext-1.0.xsd#
QMwCAG ...
</wsse:BinarySecurityToken>
</wsse:Security>
```

- La troisième option est celle d'envoyer une référence à un jeton de sécurité au lieu du jeton lui-même, ceci peut être fait en utilisant l'élément <SecurityTokenReference> qui contient une URI.

L'intégrité des messages est gérée par la spécification XML Signature en combinaison avec des jetons de sécurité. La signature renforce l'identité de l'appelant et assure une confidentialité des données ou informations transmises.

WS-Security ne propose que des options pour l'authentification, l'intégrité des messages et la confidentialité des messages, mais ne décrit pas la stratégie ou l'option utilisée à la fois par le consommateur et le fournisseur de service WEB. C'est là où intervient WS-SecurityPolicy.

1.1.3 WS-SecurityPolicy

WS-SecurityPolicy fait partie des spécifications relatives à WS-Policy.

WS-Policy se base sur un modèle de programmation déclarative qui permet d'attacher des pré-requis, des stratégies et des paramètres nécessaires à des services Web. WS-SecurityPolicy spécifie des stratégies de sécurité, il définit un ensemble de types d'assertions, elles sont décrites ci-dessous :

wsse:SecurityToken	Spécifie un type exigible du jeton de sécurité défini par WS-Security.
wsse:Integrity	Spécifie un format de signature défini par WS-Security.
wsse:Confidentiality	Spécifie un format de cryptage défini par WS-Security.
wsse:Visibility	Spécifie les portions du message qui doivent être traitées ou visibles par un intermédiaire ou un endpoint.
wsse:SecurityHeader	Spécifie l'utilisation du header Security du message.
wsse:MessageAge	Spécifie la durée maximale pour invalider les messages.

WS-SecurityPolicy permet de découpler le code de l'infrastructure de sécurité, il permet donc des stratégies de sécurité extensibles et modifiables.

Il permet aussi de réduire le code relatif à la sécurité, ce qui ajoute une flexibilité dans la gestion de la sécurité des services Web.

WS-Security est une spécification conçue afin d'être utilisé par les versions SOAP 1.1 et 1.2 de SOAP. WS-Securitypolicy peut être utilisée pour développer des applications basées sur des assertions XML associées à un endpoint d'un service Web ou un document WSDL. Elle définit un cadre pour préciser les impératifs et les stratégies d'un service Web. Elle permet aussi d'associer à un service Web des renseignements de sécurité et de routage.

La lecture des assertions WS-SecurityPolicy informe le consommateur du service sur les dispositifs de sécurité qui sont requis par le fournisseur de service, comme le format du jeton d'authentification, la nécessité de signature du message etc.

L'exemple ci-dessous précise que le fournisseur exige une intégrité du message en utilisant un algorithme de signature XML et requiert une authentification utilisant les jetons de type X.509.

```
<wsp:Policy xmlns:wsp="http://schemas.xmlsoap.org/ws/2002/12/policy"
xmlns:wssp="http://schemas.xmlsoap.org/ws/2002/12/secext">
  <wsp:All>
    <wssp:Integrity wsp:Usage="wsp:Required">
      <wssp:Algorithm Type="wssp:AlgSignature"
URI="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
    </wssp:Integrity>
  </wsp:All>
</wsp:Policy>
```

```
<wssp:SecurityToken>  
  <wssp:TokenType>wsse:X509v3</wssp:TokenType>  
</wssp:SecurityToken>  
</wsp:All>  
</wsp:Policy>
```

1.1.4 WS-Trust

Il existe plusieurs formats des jetons de sécurité (Certificats X 509, Ticket Kerberos, Assertions SAML, politiques XACML, etc), ceci nécessite de la part des acteurs d'un service Web de comprendre ces différents jetons. Par conséquent, il n'existe aucune garantie qu'un récepteur de message SOAP puisse comprendre un jeton de sécurité envoyé par un client d'un service web. Un nouveau problème se pose, celui de l'interopérabilité en terme de sécurité. Comment sera mappé un jeton de sécurité reçu de la part d'un intervenant dans le service Web ? Par quel jeton sera-t-il remplacé ? Quel type de jeton renvoyer pour qu'il soit compréhensible par le client ?

Pour répondre à ce besoin d'interopérabilité et pour installer une confiance entre les intervenants dans le service web, deux spécifications ont été introduites : WS-Trust et WS-SecurityPolicy. On se concentrera sur la première spécification : WS-Trust.

WS-Trust décrit une structure destinée à gérer, à établir et à évaluer des relations de confiance permettant aux services Web d'interopérer d'une manière sûre.

Un consommateur et un fournisseur de service peuvent communiquer en Out-Of-Band afin d'échanger les certificats de sécurité. Par exemple, un demandeur de service peut obtenir avoir besoin d'obtenir une clef publique de la part du fournisseur afin d'effectuer un cryptage des données avant d'envoyer le message. WS-Trust définit le protocole pour contrôler une relation de confiance comme celle –ci.

WS-Trust établit un protocole pour émettre, renouveler et valider les jetons de sécurité.

Le modèle de sécurité défini par WS-Trust est basé sur le processus suivant : Si un fournisseur de service web requiert que les messages provenant d'un demandeur comportent un ensemble de certificats (identifiant-mot de passe, clef, permission ...), Si le demandeur envoie un message sans ces certificats, l'échange sera refusé et une erreur sera renvoyée.

Le fournisseur spécifie ces contraintes via Ws-SecurityPolicy.

Les jetons de sécurité sont demandés en utilisant l'élément <RequestSecurityToken> et retournés via l'élément <RequestSecurityTokenResponse>.

Mécanisme à WS-Trust

WS-Trust est utilisé afin d'assurer l'interopérabilité entre les multiples jetons de sécurité qui peuvent être transportés dans un message sécurisé via WS-Security. Cette interopérabilité ne se place pas qu'au niveau syntaxique. Par exemple, un service peut accepter des jetons Kerberos mais peut ne pas comprendre des jetons générés par des centres de distribution arbitraires.

En général, un service qui reçoit un message SOAP de type WS-Security doit gérer trois problématiques principales :

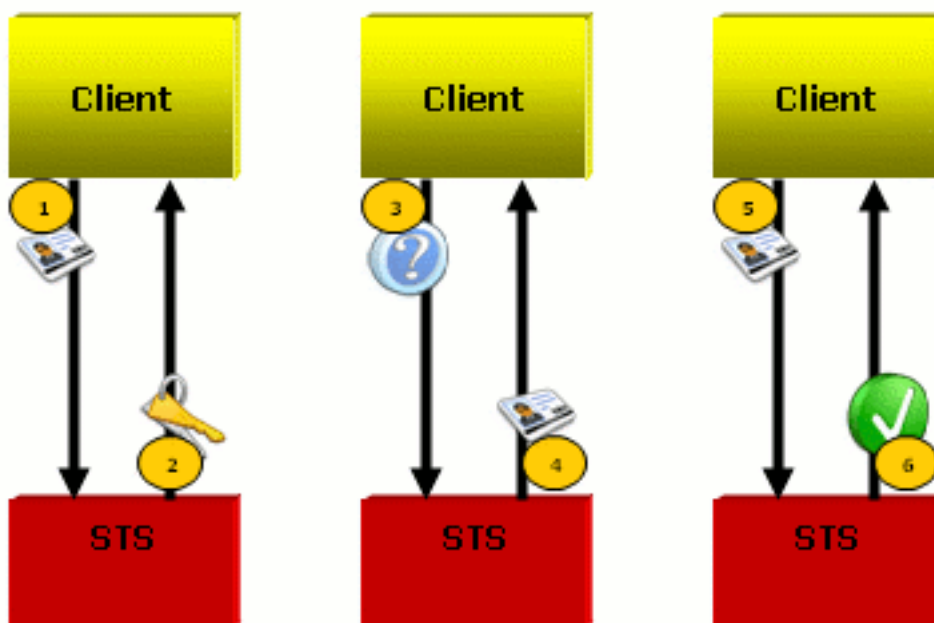
- Le format : le format peut être incompréhensible.
- La confiance : Le récepteur peut ne pas établir une chaîne de confiance.
- L'espace de nommage : Le récepteur peut avoir une syntaxe différente de celle de l'émetteur.

WS-Trust resout ces problématiques potentielles en établissant un système d'échange basé sur un protocole request/response. Le client envoie une RequestSecurityToken au STS (Security Token Service), la requête comporte le jeton de sécurité utilisé pour l'échange. Le STS répond avec une RequestSecurityTokenResponse contenant le nouveau jeton.

Les trois problématiques sont donc gérées comme suit :

- Le format : Le STS renvoie un jeton totalement compréhensible par le service récepteur.
- La confiance : Le jeton retourné a été signé par le STS ; le service récepteur s'attend à avoir en retour un jeton valide afin d'établir une relation de confiance.
- L'espace de nommage : La syntaxe correspond à celle attendue par le récepteur. En plus de l'échange des jetons de sécurité, WS-Trust permet de demander de jetons de sécurité et de les valider.

Ci-dessous, une figure représentant les différents échanges effectués entre le STS et le client du service sécurisé.



Le scénario WS-Trust

Dans une architecture basique un Client envoie un message SOAP contenant une entête WS-Security. Le but de la Gateway (passerelle) et du STS est d'assurer que le service reçoit un message sécurisé avec des jetons valides et compréhensibles. La gateway et le STS utilisent des messages WS-Trust pour permettre un traitement interopérable entre le client et le service demandé.

La gateway se trouve en frontal des services. Les clients peuvent attaquer directement la gateway qui se charge après de renvoyer au bon service, pour cela l'adresse du service dans le fichier WSDL doit pointer

directement sur l'adresse de la gateway. Le client peut aussi attaquer le service directement, dans ce cas la gateway est déclenchée à l'appel du service afin de traiter les jetons de sécurité. Le traitement exécuté par le service n'est effectué que si les jetons sont validés et échangés.

La gateway doit être capable de déduire les préférences en terme de jetons de sécurité exigés par les services appelés.

L'appel du STS effectué par la gateway est sécurisé, il convient qu'il soit effectué au niveau de la couche transport.

Afin de comprendre le but de cette architecture, prenons l'exemple ci-dessous :

1. Le client ne comprend que les certificats X.509. Il passe par un CA (Certificate Authority) afin de signer ses messages pour assurer l'intégrité des messages et prouver leur provenance.
2. Le service ne comprend que les assertions SAML.
3. Le service n'a aucun moyen lui permettant d'établir une relation de confiance avec le client.
4. Le client n'est pas supposé connaître le type de jetons attendus par le service.

La communication entre le client et le service se déroule en quatre étapes.

1. Le client SOAP envoie la requête suivante à la gateway ou au service.

```
1 <soap:Envelope>
2 <soap:Header>
3 <ws:Security>
4   <ws:BinarySecurityToken id="X509token" ValueType="X.509">
5     sdfOIDFKLSoidesdfk ...
6   </ws:BinarySecurityToken>
7   <ds:Signature>
8     <ds:Reference>
9       <ds:Ref URI="#PO"/>
10    </ds:Reference>
11    <ds:SignatureValue>akjsdfklsf</ds:SignatureValue>
12    <ds:KeyInfo>
13      <ws:BinarySecurityTokenReference URI="#X509token"/>
14    </ds:KeyInfo>
15  </ds:Signature>
16 </ws:Security>
17 </soap:Header>
18 <soap:Body>
19   <po:PurchaseOrder ID="PO"/>
20 </soap:Body>
21 </soap:Envelope>
```

La gateway intercepte le message et déduit que le client utilise des certificats X.509, sachant que les services s'attendent à recevoir des assertions SAML. Elle renvoie ce message au STS qui se chargera de mapper une assertion SAML au certificat X.509.

2. L'échange des jetons de sécurité doit se faire sans perte d'informations sur l'identité du client, ceci peut être considéré comme une information déterminante dans le traitement effectué par le service.

La gateway envoie donc la requête suivante au STS :

```

1 <soap:Envelope>
2 <soap:Header>
3 <ws:Security>
4
5 </ws:Security>
6 </soap:Header>
7 <soap:Body>
8 <wstrust:RequestSecurityToken>
9 <wstrust:TokenType>SAML</TokenType>
10 <wstrust:RequestType>ReqExchange</RequestType>
11 <wstrust:OnBehalfOf>
12 <ws:BinarySecurityToken id="originaltoken"
13 <ws:BinarySecurityToken id="originaltoken"
14 Value="sdfoIDFKLSoidefsdfk ..."
15 Value="sdfoIDFKLSoidefsdfk ..."
16 </ws:BinarySecurityToken>
17 </wstrust:OnBehalfOf>
18 </wstrust:RequestSecurityToken>
19 </soap:Body>
20 </soap:Envelope>

```

Le corps SOAP (SOAP Body) contient l'élément RequestSecurityToken définissant le jeton client et celui attendu en retour. L'élément OnBehalfOf notifie le STS que la requête envoyée par la gateway est faite sans que le service ne soit au courant.

- Une fois que la requête envoyée par la passerelle a été identifiée et authentifiée, le STS génère une assertion SAML et envoie une réponse signée à la gateway.

```

1 <soap:Envelope>
2 <soap:Header>
3 <ws:Security>
4
5 </ws:Security>
6 </soap:Header>
7 <soap:Body>
8 <wstrust:RequestSecurityTokenResponse>
9 <wstrust:TokenType>SAML</TokenType>
10 <wstrust:RequestedSecurityToken>
11 <saml:Assertion
12 <saml:Assertion
13 AssertionID="2se8e/vaskfsdif="
14 Issuer="www.sts.com"
15 IssueInstant="2002-06-19T16:58:33.173Z">
16 <saml:Conditions
17 <saml:Conditions
18 NotBefore="2002-06-19T16:53:33.173Z"
19 NotOnOrAfter="2002-06-19T17:08:33.173Z"/>
20 <saml:AuthenticationStatement
21 <saml:AuthenticationStatement
22 AuthenticationMethod="urn:oasis:names:tc:SAML:1.0:am:X.509"
23 AuthenticationInstant="2002-06-19T16:57:30.000Z">
24 <saml:Subject>
25 </saml:Subject>
26 </saml:AuthenticationStatement>
27 </wstrust:RequestedSecurityToken>
28 </wstrust:RequestSecurityTokenResponse>
29 </soap:Body>
30 </soap:Envelope>

```

```

24         <saml:NameIdentifier
      NameQualifier="service.com">Client</saml:NameIdentifier>
25         <saml:SubjectConfirmation>
26             <saml:ConfirmationMethod>
27                 urn:oasis:names:tc:SAML:1.0:cm:sender-vouches
28             </saml:ConfirmationMethod>
29         </saml:SubjectConfirmation>
30     </saml:Subject>
31 </saml:AuthenticationStatement>
32     <ds:Signature>
33     <!-- calculated by STS --> </ds:Signature>
34 </saml:Assertion>
35 </wstrust:RequestedSecurityToken>
36 </wstrust:RequestSecurityTokenResponse>
37 </soap:Body>
38 </soap:Envelope>

```

Le STS retourne un élément RequestedSecurityToken [lignes 13-34]. La signature du STS est définie dans l'élément Signature [Ligne 32]. Le STS associe aussi un identifiant client à la réponse [Ligne 24]. Ceci présume que le STS a accès au référentiel clients.

4. La passerelle reçoit le message du STS, extrait l'assertion SAML à partir de ce dernier et recrée une nouvelle version du message envoyé par le client. Ci-dessous un exemple :

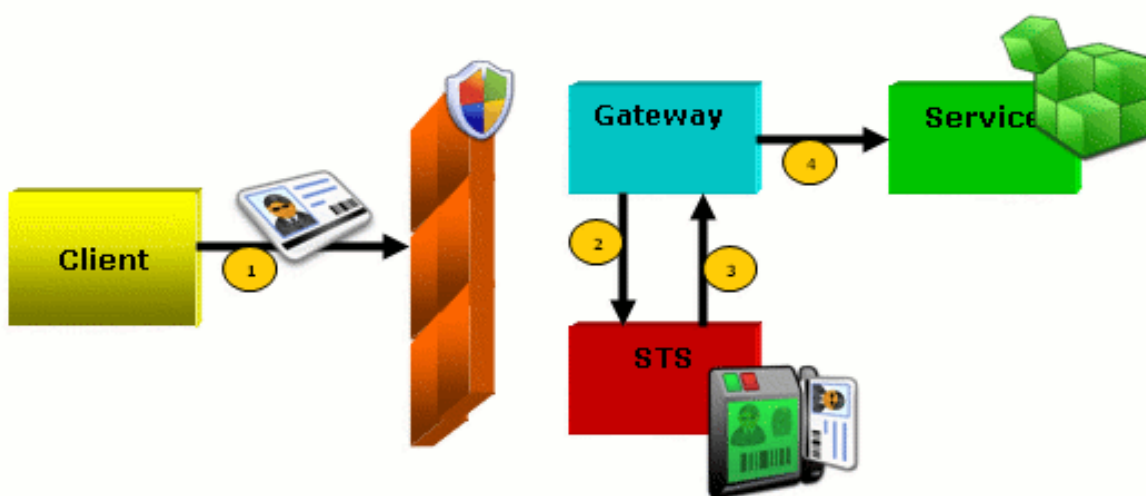
```

1 <soap:Envelope>
2 <soap:Header>
3 <ws:Security>
4     <saml:Assertion
5         AssertionID="2se8e/vaskfsdif="
6         Issuer="www.sts.com"
7         IssueInstant="2002-06-19T16:58:33.173Z">
8         <saml:Conditions
9             NotBefore="2002-06-19T16:53:33.173Z"
10            NotOnOrAfter="2002-06-19T17:08:33.173Z"/>
11         <saml:AuthenticationStatement
12             AuthenticationMethod="urn:oasis:names:tc:SAML:1.0:am:X.509"
13             AuthenticationInstant="2002-06-19T16:57:30.000Z">
14             <saml:Subject>
15                 <saml:NameIdentifier>
16                     Client</saml:NameIdentifier>
17                 <saml:SubjectConfirmation>
18                     <saml:ConfirmationMethod>
19                         urn:oasis:names:tc:SAML:1.0:cm:sender-vouches
20                     </saml:ConfirmationMethod>
21                 </saml:SubjectConfirmation>
22             </saml:Subject>
23         </saml:AuthenticationStatement>
24         <ds:Signature>
25         <!-- calculated by STS --> </ds:Signature>
26     </saml:Assertion>
27 </ws:Security>
28 </soap:Header>

```

```
27 <soap:Body>
28     <po:PurchaseOrder ID="PO" />
29 </soap:Body>
30 </soap:Envelope>
```

La passerelle a inséré l'assertion SAML dans l'entête de la requête SOAP [Lignes 4-24]. L'élément `ConfirmationMethod` [Ligne 17] permet au service d'authentifier la provenance du message. Il est intéressant de noter que la signature du STS est persistante, ce qui installe indirectement une relation de confiance entre le service et le client d'origine.



1.1.5 WS-SecureConversation

Il est possible que plusieurs nœuds ou participants à une transaction aient à effectuer une conversation impliquant des échanges multiples et fréquents. Le fait de traiter lors de chaque échange les spécificités liées à la sécurité rend la communication lourde, verbeuse et moins performante sachant que ce traitement peut être le même à chaque échange.

WS-SecureConversation est introduit afin de résoudre cette problématique. WS-SecureConversation apporte les avantages suivants :

- Performant dans le cas d'échanges multiples.
- Le contexte de sécurité est établi via WS-Trust ce qui permet de définir un profil distinct pour l'émission, le renouvellement et l'annulation des jetons de sécurité.

WS-SecureConversation est basée sur les spécifications WS-Trust et WS-Security et permet de définir un contexte de sécurité partagé par plusieurs applications. Il est donc possible que deux applications échangent plusieurs messages sans avoir à échanger des credentials (certificats ou informations de sécurité) au niveau de chaque message.

Elle définit un nouveau jeton de contexte pour le bloc <wsse:security>. Le contexte dispose d'un identifiant unique qui peut être utilisé afin de stocker temporairement ou de façon persistante des informations d'authentification, d'autorisation et de cryptage.

Les deux parts (consommateur et fournisseur du service web) effectuent une copie des certificats et n'ont plus à échanger les informations de sécurité à chaque appel SOAP.

```
<SecurityContextToken wsu:Id="...">
  <wsc:Identifiant>...</wsc:Identifiant>
</SecurityContextToken>
```

Ci-dessous un exemple parlant de l'utilisation de WS-SecureConversation

```
<s:Envelope xmlns:s='http://www.w3.org/2003/05/soap-envelope' >
  <s:Header>
    <ws:Security s:mustUnderstand='true' >
      <wsc:SecurityContextToken>
        <wsc:Identifiant>
          uuid:652d2aaa-4857-4d8c-865c-f9549e5806f0
        </wsc:Identifiant>
      </wsc:SecurityContextToken>
    </ws:Security>
  </s:Header>
</s:Envelope>
```

```
    </ws:Security>
  </s:Header>
  <s:Body wsu:Id='request'>
    ...
  </s:Body>
</s:Envelope>
```

1.1.6 WS-Federation

WS-Federation propose une généralisation des spécifications de WS-Trust en proposant des mécanismes de fédération et de gestion d'identité. La fédération d'identité consiste en la mise en place d'un système de SSO (Single Sign On), permettant de propager des identités entre les différents participants à un service et d'un système SSO (Single Sign Off ou Global Logout) qui permet de mettre fin à toutes les sessions ouvertes par les participants de la transaction. WS-Federation permet entre autres d'effectuer du profiling d'utilisateurs.

WS-Federation définit la manière avec laquelle les relations de confiance sont établies à travers des domaines de sécurité.

Le modèle de sécurité basé sur WS-Federation permet de définir un pseudonyme associé à chaque entité. Ceci est effectué via deux services, un AS (Attribute Service) et un PS (Pseudonym Service). Pour la définition de ces deux services, se référer à la terminologie relative à la sécurité SOA.

Parmi les notions utilisées par WS-Federation, on trouve :

- Security Token Service (STS)
- Identity Provider (IP)

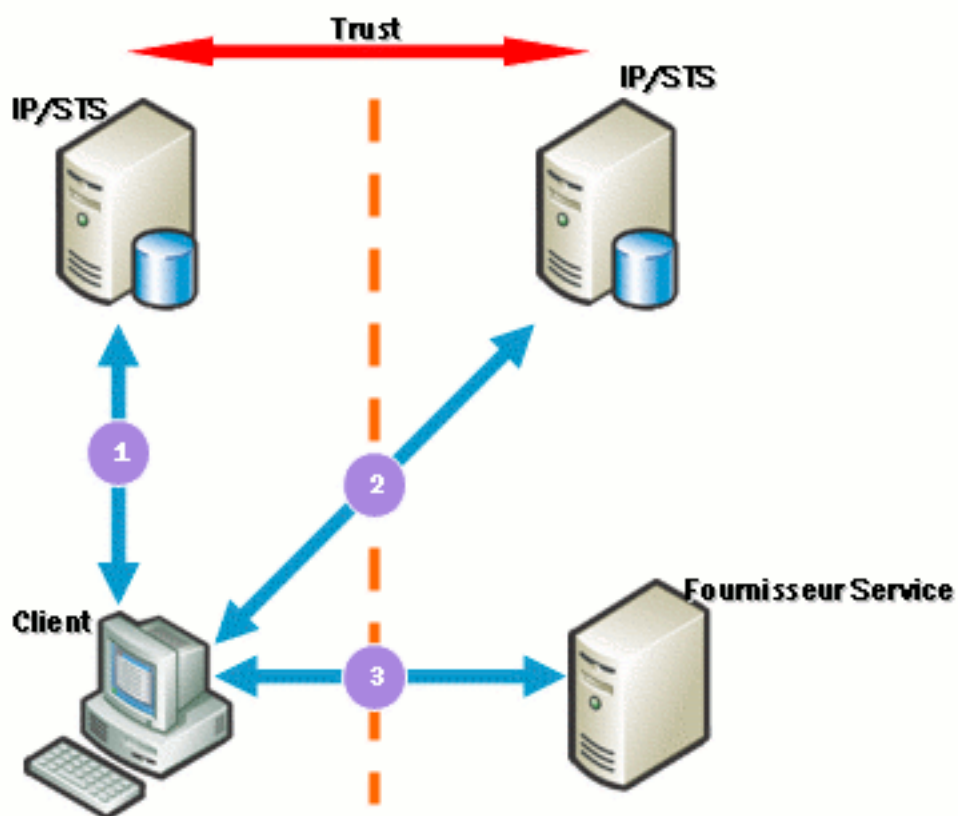
L'approche de la fédération permet de supporter les différentes topologies de confiance initiées par WS-Trust en citant quelques unes : Direct Trust, Indirect Trust, Delegation.

Ce modèle est simple à mettre en œuvre. Cependant la politique de fédération dépend du type du client appelant. Deux types ou profils sont différenciés, Un profil Actif (Client SOAP capable de lire un WSDL et comprendre le résultat de retour d'un service Web), un profil Passif (Navigateur Web classique).

L'une des problématiques qui restent à gérer en utilisant WS-Federation est celle de la protection de l'identité en assurant l'anonymat de l'identité.

L'avantage le plus intéressant qui est dû à l'utilisation de WS-Federation consiste en la possibilité de s'intégrer aux différentes infrastructures qui fournissent les informations sur les identités et qui sont capables de comprendre les différents formats de jetons de sécurité ainsi que les annuaires de services et les fabriques d'attributs d'identité.

Direct Trust



Le flux dans un contexte direct est le suivant :

1. L'infrastructure Client récupère le fichier de policy associé au service à demander.
2. Une fois le fichier de policy récupéré, l'infrastructure client demande un jeton à son STS.
3. Une fois le jeton récupéré, un appel est fait au STS de l'infrastructure du service afin de récupérer un jeton valide pour finalement appeler le service.

1.1.7 SAML

SAML (Security Assertion Markup Language) est un langage promu par l'organisation OASIS et qui supporte le système SSO et permet la propagation d'informations d'autorisation. Ceci dit un fournisseur de services web exige une seule phase d'authentification et d'autorisation à ses services web et aux services web de ses partenaires. Les partenaires doivent être capable de reconnaître l'identité de l'utilisateur.

Le profile WS-Security SAML définit l'utilisation de SAML via SOAP. SAML peut très bien être utilisé indépendamment de SOAP et de WS-Security.

SAML définit trois concepts : assertions, protocole et binding.

Les assertions correspondent à types :

- Les assertions de type Authentification,
- Les assertions de type Attributs,
- Les assertions de type Autorisation.

Le protocole définit la façon dont communiquent les applications (elles peuvent être de type service Web) avec SAML afin de procéder à l'authentification et à l'autorisation.

Les assertions SAML fournissent les informations relatives aux identités. Une assertion peut imbriquer d'autres assertions de types différents.

Les assertions SAML sont référencées par la tag `<wsse:SecurityTokenReference>` quand elles sont utilisées via WS-Security. Elles peuvent être placées directement au niveau de l'entête `<wsse:Security>`. Elles peuvent aussi avoir un numéro de version ainsi qu'une signature, définir un protocole et un comportement.

SAML permet aussi de crypter et de dater les assertions.

1.1.8 XACML

XACML (eXtensible Access Control Markup Language) est un langage qui permet de décrire des stratégies de contrôle d'accès.

Les mécanismes de contrôle d'accès se font à deux niveaux :

- La définition et la gestion des informations relatives aux contrôles d'accès,
- Le contrôle afin d'autoriser l'accès au service web,

Les spécifications XACML fournissent les méthodes pour répondre à une demande d'autorisations. Elles peuvent aussi servir à interconnecter plusieurs moteurs de stratégies de contrôle d'accès.

XACML définit aussi un ensemble de règles d'accès à des données en fonction des rôles des demandeurs par exemple.

1.1.9 XKMS

XKMS (XML Key Management Specification) est un mécanisme basé sur XML pour PKI (Public Key infrastructure).

PKI utilise une clef publique afin de crypter, décrypter, signer, autoriser et vérifier l'authenticité d'une information, y compris les messages des services web. Les clefs publiques et privées peuvent être utilisées pour le cryptage et la signature XML par exemple.

Les spécifications XKMS définissent un ensemble de services web capables de gérer les tâches d'inscription et de validation de clefs et autres détails PKI.

En d'autres termes XKMS définit des interfaces pour les services web comportant des systèmes de gestion de clefs.

XKMS marche avec tous les systèmes PKI.

1.2 Best Practices

- Les éléments timestamp et nonce doivent tous deux être signés. Dans le cas contraire, ils peuvent être facilement modifiés et ne peuvent plus empêcher les attaques de réexécution (replay)
- Il n'existe pas de méthode unique pour sécuriser une architecture Orientée Services, il convient de définir les priorités en fonction des besoins de l'entreprise – Se référer à la matrice de sécurité -.
- Prendre sérieusement en compte les spécifications WS-Security et SAML pour la mise en place de services reposant sur l'authentification et l'autorisation
- Prendre sérieusement en compte les spécifications XKMS et SAML pour les services nécessitant une protection des données contre les attaques.
- Combiner XKMS et WS-Security permet une utilisation forte de la signature digitale et des assertions SAML.
- Il convient d'utiliser la signature et le cryptage des fichiers XML lors de la combinaison de SAML et WS-Security.
- Le trafic SAML devrait être sécurisé en utilisant XKMS basé sur PKI.

1.3 Les Solutions Open Source

Il existe plusieurs projets travaillant sur l'implémentation de spécifications relatives à la sécurité des services Web. La majorité de ces implémentations utilisent les langages Java ou C++. Ci-dessous un descriptif des projets les plus conséquents :

- Apache XML Security : Ce projet a pour but de fournir une implémentation des standards de sécurité relatifs à XML. Ces standards sont XML-Signature et XML Processing. Le projet vise de travailler sur les spécifications XML Key Management (XKMS)
- OpenSAML consiste en un ensemble de bibliothèques Open Source Java et C++ qui sont en conformité avec les spécifications SAML 1.0 et 1.1.
- Apache Directory Project : Ce projet livre un module de sécurité comportant :
 - Un framework d'authentification : AuthX
 - Une librairie compatible RFC Kerberos
 - ChangePw pour changer les mots de passe de manière sécurisée.
- VeriSign WS-Security toolkit : Librairie Open Source pour faciliter l'utilisation et l'intégration de WS-Security dans le développement des applications sécurisées basées sur les services Web.
- La Crypto API de Bouncy Castle inclut les points suivants:
 - Une API légère de cryptographie en Java.
 - Un fournisseur pour la JCE et JCA.
 - Une implémentation convenable de la JCE 1.2.1.
 - Une bibliothèque pour lire et écrire des objets encodés en ASN.1.
 - Des Générateurs pour les versions 1 et 3 des certificats X.509 et des fichiers PKCS12.
 - Des Générateurs pour la version 2 des certificats de l'attribut X.509.
 - Des Générateurs/Processeurs pour S/MIME et CMS (PKCS7).
 - Des Générateurs/Processeurs pour OCSP (RFC 2560).
 - Des Générateurs/Processeurs pour TSP (RFC 3161).
 - Des Générateurs/Processeurs pour OpenPGP (RFC 2440).
 - Une version jar signée, appropriée pour les JDK 1.4/1.5 et la JCE de Sun.

2.1 Où trouver les spécifications?

SAML	http://www.oasis-open.org/committees/security/
Security Services TC	http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wss
WS-Federation	http://www-106.ibm.com/developerworks/webservices/library/ws-fedworld/
WS-Security	http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wss
WS-SecureConversation	http://www-106.ibm.com/developerworks/webservices/library/ws-secon/
WS-SecurityPolicy	http://www-106.ibm.com/developerworks/webservices/library/ws-secpol/
WS-Trust	http://msdn.microsoft.com/library/en-us/dnglobspec/html/ws-trust.asp
XML-Encryption	http://www.w3c.org/Encryption/2001/
XML-Signature	http://www.w3c.org/Signature/

2.2 Ressources de documentation

Ci-dessous des éléments de documentation utiles pour aborder la sécurisation des Services Web.

Basic Security Profile Working Group	http://www.ws-i.org/deliverables/workinggroup.aspx?wg=basicsecurity
Public Key Infrastructure (PKI) (Anglais)	http://www.pki-page.org/
Public Key Infrastructure (PKI) (Français)	http://www.hsc.fr/ressources/cours/pki/index.html.fr
WS-Security Kerberos	http://www.oasis-open.org/committees/download.php/1049/WSS-Kerberos-03.pdf
SAML (Security Assertion Markup Language)	http://www.oasis-open.org/committees/download.php/1048/WSS-SAML-06.pdf
REL (Rights Express Language)	http://www.oasis-open.org/committees/download.php/7347/oasis-____-wss-REL-token-profile-1.0-draft08-clean.pdf
OpenSAML 1.0.1 - an Open Source Security Assertion Markup Language implementation	http://www.opensaml.org/
The XML Apache Security Project	http://xml.apache.org/security/index.html
Ehe Apache Directory Project - Kerberos	http://directory.apache.org/subprojects/kerberos.html
La légion de Bouncy Castle	http://www.bouncycastle.org/fr/index.html
AXIS WSSE Security	http://axis-wsse.sourceforge.net/#home
VeriSign Offers Open Source WS-Security Implementation and Integration Toolkit	http://www.verisign.com/verisign-inc/news-and-events/news-archive/us-news-2002/page_000810.html
FIX : Financial Information eXchange protocol	http://www.fixprotocol.org
IIOP : Internet Inter-ORB Protocol	http://www.omg.org
UDDI : Universal Description, Discovery and Integration	http://www.uddi.org