MASTER EN SCIENCES INFORMATIQUES

Sécurité des API Web

Soumaila, Ramadan

Award date:
2017

Awarding institution:
Universite de Namur

Université de Namur
Faculté d'informatique
Année académique 2016–2017

Sécurité des API Web

Ramadan Soumaila

Promoteur : _____ (Signature pour approbation du dépôt - REE art. 40)
Pr. Philippe Thiran

Co-promoteur : M. Bojan Spasic

Mémoire présenté en vue de l'obtention du grade de
Master en Sciences Informatiques.

Résumé

Les API Web constituent le point central des infrastructures telles que les systèmes IoT, les plate-formes en ligne des entreprises commerciales et bien d'autres. Ces API manipulent des données qui peuvent être sensibles, il est alors nécessaire de les sécuriser. Pour cela, de nombreux outils existent, mais trouver ou savoir les plus avancés et adéquats à certains contextes n'est pas toujours facile. Ce travail consiste à mettre en lumière le concept de sécurité des API. Nous procédons à une étude des techniques de sécurité des API les plus avancées, puis nous les comparons par la suite, et finalement nous développons un système qui montre leur utilisation.

Mots clés : *Sécurité*, *API*, *Web*.

Abstract

The Web API constitutes the central point of infrastructures such as IoT systems, commercial organization on-line platforms and so on. These API treat data which can be sensitive, it is then necessary to secure them. For that purpose, numerous tools exist, but to find or to know the most advanced and adequate to certain contexts is not always easy. This work consists in highlighting the concept of security of API. We proceed to a study of the techniques security of the most advanced API, then we compare afterward, and finally we let us develop a system which shows their use.

Key words: *Web*, *API*, *Security*.

Remerciements

J’exprime ma vive gratitude aux personnes physiques et institutions qui ont contribué à rendre meilleur mon travail, notamment :

- Prof Phillipe Thiran mon promoteur ;
- M. Bojan Spasic mon co-promoteur ;

Que tous ceux qui m’ont aidé, de près ou de loin, trouvent ici l’expression de mes sentiments les meilleurs.

Table des matières

Résumé/Abstract **2**

Remerciements **3**

Glossaire **7**

Introduction **11**

1 WEB API **13**
1.1 Les Web Services 13
1.1.1 Simple Object Access Protocol (SOAP) 14
1.1.2 Web Service Description Language (WSDL) 16
1.1.3 Universal Description, Discovery and Integration (UDDI) 18
1.2 Les Restful Services 19
1.3 Comparaison approche Web Service vs approche Restful Services 21

2 La Sécurité dans les API **23**
2.1 Les objectifs de sécurité dans les API 23
2.1.1 Authentification et Autorisation 24
2.1.2 Confidentialité, Intégrité, Non-répudiation 25
2.2 Les niveaux de sécurité 25
2.2.1 Transport Level Security 25
2.2.2 Message Level Security 26

3 La Sécurité dans les Web Services SOAP **27**
3.1 Security Assertion Markup Language (SAML) 27
3.2 Web Service Security (WSS) 29
3.3 Web Service Policy (WS-Policy) 31
3.4 Web Service Trust 32
3.5 Web Service Federation 33
3.6 Web Service SecureConversation 34
3.7 Relations entre les standards de sécurité des services Web SOAP 35

4 La sécurité dans les API Restful **37**
4.1 Quelques efforts message level secuity 37
4.2 Quelques Outils libres de sécurité 38
4.2.1 Transport Layer Security (TLS)/ Secure Socket Layer (SSL) 38
4.2.2 OAuth 2.0 39
4.2.3 OpenID Connect 44
4.2.4 JavaScript Object Signing and Encryption (JOSE) 50

5 Analyse Comparative Fonctionnelle **53**
5.1 Comparaison des solutions d'authentification et/ou d'autorisation 53
5.1.1 OAuth 2.0 vs OpenID Connect 53
5.1.2 OpenID Connect vs SAML 54
5.1.3 OAuth 2.0 vs SAML 55
5.1.4 Tableau comparatif 56
5.2 Comparaison des solutions de confidentialité et d'intégrité 57
5.2.1 TLS/SSL vs WS-Security 58
5.2.2 TLS/SSL vs JSON Object Signature & Encryption 59
5.2.3 WS-Security vs JSON Object Signature & Encryption 59
5.2.4 Tableau comparatif 60

6 Exemple d'application **62**
6.1 Scénario 62
6.2 Résultats de l'implémentation 63

Conclusion **69**

A Les 2 autres flows de OAuth 2.0 **73**
A.1 Resource Owner Password Credentials flow 73
A.2 Client Credentials flow 74

Table des figures

1.1 Exemple de Requête SOAP 15
1.2 Exemple de Réponse SOAP 15
1.3 Exemple de WSDL 17
1.4 Processus de découverte et d'inscription à l'UDDI 18
1.5 Correspondance entre les opérations REST, CRUD et SQL 20

2.1 Scénario de Web Service sans sécurité 24

3.1 Protocole SAML 28
3.2 Exeampe de message SOAP Sécurisé 30
3.3 Scénario de Web Service sans sécurité 31
3.4 Scénario de Web Service sans sécurité 31
3.5 Déroulement scénario WS-Trust 33
3.6 Relations entre standard de sécurité SOAP 35

4.1 OAuth Abstract Protocol Flow 40
4.2 OAuth Client Registration 41
4.3 Authorization Code Flow 42
4.4 OAuth Implict Flow 44
4.5 OpenID Connect Abstract Protocol Flow 45
4.6 Enrégistrement dynamique du client OpenID Connect 46
4.7 OpenID Connect Code Authorization flow 48
4.8 OpenID Connect Implicit flow 49
4.9 OpenID Hybrid flow 50
4.10 Exemple de clé JWK 51
4.11 Exemple de JWS 51
4.12 Données en BASE64URL d'un JWS 51
4.13 Exemple de JWE 52
4.14 Données en BASE64URL d'un JWE 52

6.1 Enrégistrement du client sur la console de Google 63
6.2 Crédentials du client 64
6.3 Fenêtre d'Authentification du client 65
6.4 Authentification de l'utilisateur sur Google 66
6.5 Autorisation de l'utilisateur 67
6.6 Accueil application cliente 68

A.1 OAuth Resource Owner Password Credentials flow 74
A.2 OAuth Client Credentials flow 75

Liste des tableaux

1.1 Tableau comparative service SOAP vs services REST 22

4.1 Les flows d'authentification OpenID Connect 45

5.1 Tableau comparatif des solutions d'autorisation et d'authentification 57
5.2 Tableau comparatif des solutions de confidentialité et d'intégrité 60

Glossaire

AS :

Autorization Server

CORBA :

Common Object Request Broker Architecture

FTP :

File Transfer Protocol

HTTP :

Hyper Text Transfer Protocol

HTTPS :

Hyper Text Transfer Protocol Secure

IdP :

Identity Proider

JOSE :

JSON Object Signature and Encryption

JSON :

Javascript Objet Notation

OASIS :

Organization for the Advancement of Structured Information Standards

REST :

Representational State Transfer

RMI :

Remote Method Invocation

RPC :

Remote Procedure Call

RTS :

RequestSecurityToken

RTSR :

RequestSecurityTokenResponse

SAML :

Security Assertion Markup Language

SMTP :

Simple Mail Transfer Protocol

SOAP :

Simple Object Access Protocol

SP :

Service Provider

SSL :

Secure Socket Layer

STS :

Security Token Service

TCP/IP :

Transmission Control Protocol / Internet Protocol

TLS :

Transport Layer Security

UDDI :

Universal Description Discovery Integration

URI :

Uniform Resource Identifier

W3C :

World Wide Web Consortium

WSDL :

Web Service Description Language

XML :

Extensible Markup Language

Introduction

La technologie des API Web est un moyen rapide de distribution de l'information entre clients, fournisseurs, partenaires commerciaux et leurs différentes plates-formes. Cette dernière comme d'autres telles que les technologies RMI, DCOM et CORBA qui lui ont précédé est une instanciation du style architecturale "Service Oriented Architecture" (SOA).

De nos jours, cette nouvelle approche est catégorisée en deux types : les Web Services et les Resful Services. Conçus par les fournisseurs de logiciels tels que Microsoft, IBM et d'autres, les Web Services sont le premier type d'API Web, ils offrent l'échange de données, et assurent l'interopérabilité entre les différentes langages de programmation en utilisant comme base les technologies web connues telles que XML et HTTP pour la définition et l'échange de données. Les Restful Services sont des API Web basées sur le style "Representional State Transfer" (REST) qui définit un ensemble de contraintes à respecter pour accéder et manipuler des représentations textuelles de ressources Web.

Ces API Web de type SOAP ou REST, qui échanges des données pouvant être sensibles constituent la base de connexion des systèmes IoT, ainsi qu'un élément clé d'une économie qui transforme les organisations commerciales en plates-formes en ligne. Ainsi il est primordial que ces API soit robustes, disponibles et sécurisées, tout en permettant une intégration facile dans les systèmes existants et l'interopérabilité.

Pour répondre à ce besoin de sécurité des API Web, de nombreux outils permettant de répondre à différents aspects de sécuriser existent. Ces derniers tous autant diversifiés par leur cadre d'utilisation, leurs forces et faiblesses constituent un éventail de ressources sur internet, il est alors raisonnable d'en connaître les plus adéquats pour mieux sécuriser les api

Ainsi notre travail consiste à étudier les techniques les plus avancées qui permettent d'assurer la sécurité des API Web, de les comparer et de montrer leur utilisation dans un système.

Le présent mémoire fait le point de nos travaux et comporte six (6) chapitres. Nous commençons dans le premier chapitre par présenter les API Web. Nous abordons d'abord les Web services en présentant leurs caractéristiques et les standard nécessaire à leur utilisation. Ensuite nous passons en revue les Restful Services en mettant en évidences les contraintes du style REST. Enfin nous faisons une brève comparaison entre les Web Services et les Restful Services.

Dans le deuxième chapitre nous nous concentrons sur le concept de sécurité des API. Nous présentons premièrement les différents objectifs de sécurité nécessaire pour les API Web. Deuxièmement nous abordons les différents niveaux auxquels ces objectifs de sécurité peuvent être appliqués.

Une fois ces deux standards d'API Web, et les objectifs de sécurité des API présentés, nous examinons dans le chapitre 3 les différentes méthodes de sécurité définies pour la sécurité des services de type SOAP que l'on peut trouver dans la littérature scientifique et nous mettons en évidence les relations qui existent entre ces dernières.

Dans le chapitre 4 nous abordons la sécurité des Restful Services , nous présentons d'une part quelques efforts fournis afin de répondre a leur besoin de sécurité au niveau message et d'autre part nous abordons les outils libres de sécurité utilisés pour les protéger en mettant un accent sur leurs caractéristiques et leur fonctionnement.

Le Chapitre 5 fait l'objet d'une analyse comparative de quelques outils de sécurité de ces API Web présentés dans les chapitres 3 et 4. Nous comparons deux à deux les outils permettant de fournir les mêmes aspects de sécurité. En se basant sur les informations obtenues, nous mettons en évidences les similitudes et différences des méthodes offrant l'authentification/autorisation d'une part et d'autre part celles concernant la confidentialité et l'intégrité.Nous mettons l'accent sur les caractéristiques fonctionnelles de ces outils

Enfin le chapitre 6 montre l'utilisation de certains de ces outils de sécurité des API Web à travers un système de Restful Service que nous développons en guise d'exemple d'application.

Chapitre 1

WEB API

Dans ce chapitre nous allons présenter les différents type d'approches permettant d'implémenter des API. Après avoir défini et expliqué le fonctionnement des Web Services d'une part, Nous parlerons d'autre part de l'approche concernant les Restful Services, en la définissant, puis en présentant ses caractéristiques. Nous terminerons alors ce chapitre par une comparaison des deux approches suivi d'une petite conclusion.

1.1 Les Web Services

La World Wide Consortium (W3C) définit un Web Service comme "un système logiciel identifié par une URI, dont les interfaces et les liaisons (bindings) publiques sont définies et décrites en utilisant le langage XML. Ces dernières peuvent être découvertes par d'autres systèmes, pouvant ainsi interagir avec le Web Service grâce à sa définition. Ces interactions se font par des messages basés sur du XML dont le transport est assuré par des protocoles Internet" [1]. Selon Robert-Jan [3] "un Web Service est une application exécutée sur un système informatique auquel on peut accéder à partir d'un réseau. Il peut effectuer n'importe quel nombre d'opérations et être accessible à l'aide de messages décrits en XML ou un protocole XML tel que SOAP".

Les Web Services basés SOAP sont formés sur trois spécifications, chacune définissant son propre domaine d'utilisation [5]. Le protocole SOAP pour "Simple Object Access Protocol" est la première spécification, décrit en XML elle permet l'échange de messages [3] . Aussi décrite en XML, la seconde spécification est le WSDL "Web Service Description Language", elle fournit les informations sur le réseau de services, leurs exigences et valeurs de retour [3] . Et pour finir la troisième spécification est le UDDI "Universal Description, Discovery and Integration " décrite aussi en XML comme les deux autres, elle fournit un moyen de publier, trouver et lier les descriptions de Web Services [3].

Pour l'envoi et la réception des messages, les Web Services peuvent utiliser des protocoles d'internet tels que TCP/IP, SMTP, FTP ou encore HTTP qui est le plus souvent utilisé.

Dû au fait que leur communication implique l'exécution de procédures se trouvant sur une autre machine [3], les Web Services peuvent être considérés comme une catégorie de RPC. Le client initie la communication en envoyant une requête au service hébergé sur le serveur. Le serveur traite alors cette requête et renvoie une réponse au client.

Il existe plusieurs spécifications additionnelles qui étendent les capacités des Web Services connu sous l'appellation de "WS-*". Certaines de ces spécifications additionnelles rendent en quelques sorte les Web Services compatibles aux autres catégories de middleware. Par exemple WS-AtomicTranscation extension du protocole SOAP offre le déroulement du protocole en deux phases et permet les transactions distribuées entre plusieurs parties, remplissant ainsi l'une des principales exigences relatives au middleware orienté transaction [22].

1.1.1 Simple Object Access Protocol (SOAP)

SOAP abréviation de "Simple Object Access Protocol", est un protocole décrit en XML pour l'échange de messages entre systèmes sans avoir à se soucier du type de systèmes d'exploitations ou encore de l'environnement dans lequel évoluent ces différents systèmes [3]. En d'autres termes ce protocole définit par le W3C est indépendant de la plate forme. C'est le protocole standard utilisé par les Web Services.

Le projet de réalisation de ce protocole débute par la collaboration en 1998 des développeurs de Microsoft, de Userland Software et de DevelopMentor Incorporated. D'autres compagnies telles que IBM s'ajoutent plus tard pour contribuer à la réalisation de la version 1.1 [22].

Un message SOAP consiste en une enveloppe constituée de deux sous-éléments, une entête "header" et un corps "body". L'élément Header est optionnel et est inclus dans la spécification pour rendre possible l'ajout d'informations ou méta-données à propos contenu du message[3], comme par exemple les mécanismes de sécurité utilisés pour sécuriser le message. L'élément Body (obligatoire) contient le message en lui-même, lequel est soit une opération décrite en XML ou un type XML de retour. S'il s'agit d'un message de requête à partir d'un client, le message contiendra le nom de l'opération et ses paramètres si celles-ci sont requises. Du côté serveur, l'opération sera mappée à une méthode exécutable puis exécutée. Après son exécution, le service créera un message de réponse SOAP contenant Le type de retour de l'opération.

```
POST /Service/Soap11 HTTP/1.1
Content-Type: text/xml; charset=utf-8
SOAPAction: "urn:ICustomerService/GetListOfCustomersByName"
Host: ana2
Content-Length: 163
Expect: 100-continue

<s:Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/">
        <s:Body>
                <GetListOfCustomersByName>
                        <name>Ola</name>
                </GetListOfCustomersByName>
        </s:Body>
</s:Envelope>
```

FIGURE 1.1 – Exemple de Requête SOAP (Source : [22])

```
HTTP/1.1 200 OK
Content-Length: 693
Content-Type: text/xml; charset=utf-8
Server: Microsoft-HTTPAPI/1.0
Date: Fri, 01 Apr 2011 13:49:41 GMT

<s:Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/">
 <s:Header>
  <ActivityId CorrelationId="26ab7785-dd9f-432a-a57a-7a29d0270a3e"
Xmlns="http://schemas.microsoft.com/2004/09/ServiceModel/Diagnostics">
        353d7ea9-7603-4ecc-a1e7-53d3e9338f66
  </ActivityId>
 </s:Header>
 <s:Body>
  <GetListOfCustomersByNameResponse>
   <GetListOfCustomersByNameResult xmlns:i="http://www.w3.org/2001/XMLSchema-instance">
    <Customer>
     <Id>14</Id>
     <LastName>Normann</LastName>
     <Name>Ola</Name>
     <Address>
      <Id>0</Id>
      <PostalCode>1567</PostalCode>
      <Street>Christian Michelsens gate 6</Street>
      <City>OSLO</City>
     </Address>
    </Customer>
   </GetListOfCustomersByNameResult>
  </GetListOfCustomersByNameResponse>
 </s:Body>
</s:Envelope>
```

FIGURE 1.2 – Exemple de Réponse SOAP (Source : [22])

Les deux figures ci-dessus démontrent respectivement une requête SOAP et une réponse SOAP utilisant comme protocole de transport le HTTP. Les messages SOAP sont englobés dans l'élément Enveloppe et font partie du contenu HTTP dont l'entête précède l'enveloppe SOAP. La requête SOAP spécifie une opération nommée "GetCustomerByName" avec un paramètre "Name" dont la valeur est "Ola". La localisation du service est trouvée en concaténant les valeurs des propriétés de l'entête que sont le "Host" et le "SOAPAction", donnant dans notre cas "http ://ana2/ICustomerService/GetListOfCustomersByName". La réponse SOAP quant à elle contient des objets "Customer" dont chacun est constitué des attributs tels que le "name", le "last name" et un object "Address" contenant les éléments "street name", "postal code" et "city name".

1.1.2 Web Service Description Language (WSDL)

Lorsqu'une application souhaite publier un service sur le monde extérieur, les utilisateurs de ces services doivent savoir exactement comment les invoquer. Le WSDL qui est aussi une recommandation du W3C a été conçu pour cet objectif. Décrit dans un format XML spécifique il fournit la description détaillée des Web Services. Le WSDL définit comment SOAP peut être utilisé avec les protocoles tels que SMTP[3], il décrit les opérations qui sont disponibles ainsi que leurs paramètres et type de retours au clients. Ainsi un client ayant pris connaissance du contenu du WSDL sait comment contacter le service et quoi attendre de ce dernier. Il faut noter que le WSDL est optionnel et n'est pas requis si le client sait comment envoyer et recevoir des messages aux web service ou que la définition du Web Service soit documenté autrement.

Le WSDL définit 4 opérations basiques à savoir [3] :

- Messagerie unidirectionnel "one-way messaging", le client envoie des informations au web service sans recevoir de message en retour.
- Notification, c'est l'inverse du "one-way messaging" où le web service envoie des informations au client sans recevoir de message en retour.
- "Request/Response messaging", c'est la plus utilisée, aussi connu comme "Remote Procedure Call (RPC), un client fait une requête au service qui lui renvoie une réponse. Le WSDL définit la structure et la syntaxe des requêtes et des réponses.
- "Solicit/Response", c'est l'inverse du "Request/Reponse", le service envoie des informations au client qui lui renvoi de réponse. Le WSDL définit aussi le format des messages.

```
<?xml version="1.0" encoding="utf-8"?>
<wsdl:definitions targetNamespace="" xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
        xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:wsu="http://docs.oasis-
        open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd"
        xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/" >
  <wsdl:types>
    <xsd:schema targetNamespace="/Imports">
      <xsd:import schemaLocation="http://ana2/Service?xsd=xsd0"/>
      <xsd:import schemaLocation="http://ana2/Service?xsd=xsd1"
            namespace="http://schemas.microsoft.com/2003/10/Serialization/"/>
    </xsd:schema>
  </wsdl:types>
  <wsdl:message name="ICustomerService_GetListOfCustomersByName_InputMessage">
    <wsdl:part name="parameters" element="GetListOfCustomersByName"/>
  </wsdl:message>
  <wsdl:message name="ICustomerService_GetListOfCustomersByName_OutputMessage">
    <wsdl:part name="parameters" element="GetListOfCustomersByNameResponse"/>
  </wsdl:message>
  <wsdl:message name="ICustomerService_InsertCustomer_InputMessage">
    <wsdl:part name="parameters" element="InsertCustomer"/>
  </wsdl:message>
  <wsdl:message name="ICustomerService_InsertCustomer_OutputMessage">
    <wsdl:part name="parameters" element="InsertCustomerResponse"/>
  </wsdl:message>
  <wsdl:message name="ICustomerService_Version_InputMessage">
    <wsdl:part name="parameters" element="Version"/>
  </wsdl:message>
  <wsdl:message name="ICustomerService_Version_OutputMessage">
    <wsdl:part name="parameters" element="VersionResponse"/>
  </wsdl:message>
  <wsdl:portType name="ICustomerService">
    <wsdl:operation name="GetListOfCustomersByName">
      <wsdl:input wsaw:Action="urn:ICustomerService/GetListOfCustomersByName"
            message="ICustomerService_GetListOfCustomersByName_InputMessage"/>
      <wsdl:output wsaw:Action="urn:ICustomerService/GetListOfCustomersByNameResponse"
            message="ICustomerService_GetListOfCustomersByName_OutputMessage"/>
    </wsdl:operation>
    <wsdl:operation name="InsertCustomer">
      <wsdl:input wsaw:Action="urn:ICustomerService/InsertCustomer"
            message="ICustomerService_InsertCustomer_InputMessage"/>
      <wsdl:output wsaw:Action="urn:ICustomerService/InsertCustomerResponse"
            message="ICustomerService_InsertCustomer_OutputMessage"/>
    </wsdl:operation>
    <wsdl:operation name="Version">
      <wsdl:input wsaw:Action="urn:ICustomerService/Version"
            message="ICustomerService_Version_InputMessage"/>
      <wsdl:output wsaw:Action="urn:ICustomerService/VersionResponse"
            message="ICustomerService_Version_OutputMessage"/>
    </wsdl:operation>
  </wsdl:portType>
</wsdl:definitions>
```

FIGURE 1.3 – Exemple de WSDL (Source : [22])

1.1.3 Universal Description, Discovery and Integration (UDDI)

L'UDDI est un standard de L'OASIS qui définit un moyen de publier et de trouver les informations à propos des Web Services. C'est un produit logiciel qui offre un répertoire "repository" pour les descriptions de services. L'UDDI d'un Web Service est lancé sur le repository des UDDI et est responsable de l'inscription et de la découverte des descriptions des services. Il faut noter que l'UDDI peut être utilisé pour gérer d'autres type de descriptions autre que du WSDL. Ainsi toutes les descriptions de services publiées dans l'UDDI sont mappées aux définitions de description UDDI. L'UDDI tModel est un exemple de telle définition [22].

Pour l'inscription d'une description d'un Web Service, un service provider envoie le WSDL à l'UDDI en utilisant l'UDDI du Web Service. Le service de l'UDDI mappera ensuite ce WSDL à la description tModel avant de la stocker dans le repository UUDI.

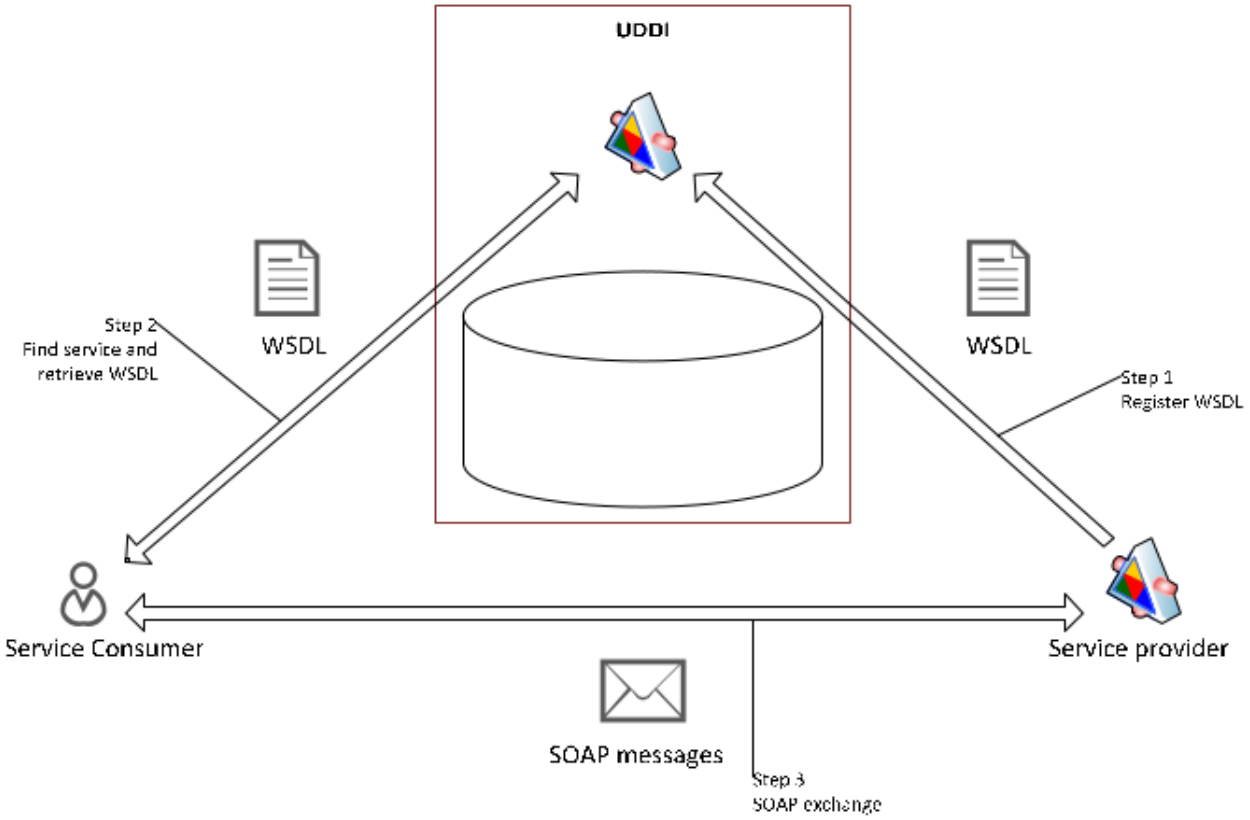

FIGURE 1.4 – Processus de découverte et d'inscription à l'UDDI(Source : [22])

La figure ci-dessus montre l'ensemble du processus permettant au public d'avoir accès au WSDL d'un Web Service. Premièrement le service provider publie son WSDL au registre de

l'UDDI complété par des propriétés décrivant à la fois le web service et le service provider comme par exemple le type de métier, le nom de l'entreprise etc. Le client peut alors fournir certains critères permettant à l'UDDI de retrouver le service, et de lui renvoyer l'URI du WSDL correspondant dans un message SOAP. Le client peut alors extraire l'URI du WSDL du message SOAP reçu et trouver le WSDL sur le réseau. Ainsi le client est en possession des informations nécessaires sur le service et est capable d'établir une connexion avec ce dernier.

1.2 Les Restful Services

Les Restful Services sont les API Web basées sur les principes REST. REST pour "Representational State Transfer", est un "ensemble coordonné de contraintes architecturales qui tente de minimiser la latence et la communication en réseau tout en maximisant l'indépendance et l'évolutivité des implémentations de composants" [7]. C'est un style architecturale pour implémenter des services sur le web en utilisant du HTTP.

Cette approches se focalise sur l'état des ressources plutôt que des messages ou des procédures [17]. REST se concentre sur les ressources d'un système, y compris la façon dont les états des ressources sont traités et transférés sur du HTTP [18].

Dans REST toute chose est considérée comme ressource, selon Dr. Roy Fielding "une ressource est quelque chose qui est assez importante pour être référencée comme une chose en soi" [7], comme exemple un fichier stocké dans une machine, un logiciel, un attribut dans une table de base de données ou un objet physique tel qu'un fruit. Les ressources peuvent être statiques ou dynamiques.

L'idée principale de REST consiste à bien exploiter le HTTP pour transférer des données entre machines, plutôt que d'utiliser un protocole qui fonctionne au-dessus la couche HTTP pour les transferts de messages. Une application conçue selon les principes REST utiliserait le HTTP pour faire des appels entre les machines, plutôt que de s'appuyer sur des mécanismes complexes comme CORBA (Common Object Request Broker Architecture), RPC (Remote Procedure Call),ou SOAP. Ainsi, les applications REST utilisent des fonctions de requêtes HTTP pour poster des données, lire des données et supprimer des données, en utilisant les fonctionnalités complètes des opérations HTTP CRUD (Créer, Lire, Mettre à jour et Supprimer). REST peut également fonctionner sur du HTTPS, fournissant la transmission sécurisée de données.

CRUD Operations	REST Key words (HTTP)	SQL (Database) Operators
CREATE – Create or add new entries	POST	INSERT
UPDATE –Update or edit existing data	PUT	UPDATE
READ – Read, retrieve	GET	SELECT
DELETE – Delete existing data	DELETE	DELETE/DROP

FIGURE 1.5 – Correspondance entre les opérations REST, CRUD et SQL (Source : [17])

Comme dit plus haut, REST est un style qui définit un ensemble de principes servant de guide pour la réalisation de services, il s'agit précisément de 6 contraintes que sont :

- Client/Server :
  Cette contrainte stipule que l'architecture doit respecter le style client/server, donc séparer les responsabilités entre le client et serveur, ainsi l'interface utilisateur est séparée de celle du stockage des données. Cela permet aux deux d'évoluer indépendamment.

- Stateless :
  La communication doit être par nature sans état, comme dans le modèle "client/serveur sans état". Ainsi chaque requête du client vers le serveur doit contenir toutes les informations nécessaires pour que cette demande soit comprise, et elle ne peut tirer profit d'aucun contexte stocké sur le serveur. L'état de la session est donc entièrement détenu par le client.

- Cacheable (mise en cache) :
  Chaque réponse d'un serveur contient un marquage explicite ou implicite de la possibilité ou non de la mettre en cache côté client ou côté serveur. Si une réponse peut être mise en cache, elle peut être réutilisée pour des requêtes ultérieures équivalentes.Cette contrainte a l'avantage d'offrir la possibilité d'éliminer tout ou une partie de certaines interactions, améliorant ainsi l'efficacité, la montée en charge et la perception de performance qu'a l'utilisateur, en réduisant la latence moyenne dans une série d'interactions. Le compromis est qu'un cache peut diminuer la fiabilité si les données obsolètes qui y sont présentes diffèrent de manière significative des données qui auraient été obtenues par une requête envoyée directement au serveur.

- Uniform Interface (interface uniforme) :
  Afin d'obtenir une interface uniforme, de multiples contraintes d'architecture sont nécessaires pour guider le comportement des composants. REST est défini par quatre contraintes d'interface :
  - identification des ressources : chaque ressource possède un identifiant.
  - manipulation des ressources par des représentations : les ressources ont des représentations définies.
  - messages auto-descriptifs : les messages expliquent leur nature. Par exemple, si une représentation en HTML est encodée en UTF-8, le message contient l'information nécessaire pour dire que c'est le cas.
  - l'hypermédia comme moteur de l'état de l'application : chaque accès aux états suivants de l'application est décrit dans le message courant. Cela permet une plus grande simplicité de l'architecture globale, une meilleure visibilité des interactions entre les composants, un découplage des mises en œuvres et des services qu'elles fournissent

- Layered System (système en couche) :
  Chaque composant voit uniquement les composants de la couche avec laquelle il interagit directement. Cela permet une limitation de la complexité de l'architecture globale, une plus grande indépendance des couches, un renforcement de la sécurité et un équilibrage des charges sur les services.

- Code on Demand (Code à la demande) :
  Cette contrainte optionnelle permet l'extension des fonctionnalités d'un client par le biais de téléchargement et d'exécution de code sous forme d'applet ou de scripts. Cela simplifie les clients en réduisant le nombre de fonctionnalités qu'ils doivent mettre en œuvre par défaut. La possibilité de télécharger des fonctionnalités après le déploiement améliore l'extensibilité du système. Elle réduit cependant la visibilité.

Compte tenu de la facilité de conception et de la flexibilité dans le codage fourni par REST, il est progressivement devenu populaire. Yahoo et eBay ont été les premiers à utiliser REST pour concevoir leurs Services Web. Ils ont ensuite été rejoints par des entreprises populaires comme Amazon et Google [17].

1.3 Comparaison approche Web Service vs approche Restful Services

En comparaison nous dirons que les Web Services et les Restful Services sont deux solutions pour un même problème,en occurrence comment accéder à des services Web. Mais les deux se

base sur deux approches différentes, les Web services se basent sur SOAP qui est un protocole tandis que les Restful Services se base sur REST qui est un style d'architecture. l'approche SOAP permet grâce au descripteur d'identifier la liste des services et objets exposés, alors que REST est sans état, ce qui lui permet d'avoir une plus grande indépendance entre le client et le serveur. SOAP s'adapte à différents protocoles de transport en évitant les problèmes de communication alors que REST n'utilise que le protocole HTTP. REST utilise l'URI comme représentation de ses ressources avec sa flexibilité d'évolution alors qu'on voit un fort couplage client / Serveur côté SOAP.

TABLE 1.1 – Tableau comparative service SOAP vs services REST

Web API	**Web Services**	**Restful Services**
Approches	SOAP (protocole)	REST (style architectural)
Identification	WSDL, UDDI	Sans état
Transport	HTTP, FTP, SMTP etc..	HTTP
Format	XML	multi-format (XML, JSON, ...)

Dans ce chapitre nous avons présenté les différentes approches permettant d'implémenter les API Web, d'une part nous avons mis en évidences les caractéristiques des API basées sur l'utilisation de SOAP et d'autre part les Restful services, et pour finir nous avons présenté une brève comparaison de l'approche SOAP à l'approche REST.

Chapitre 2

La Sécurité dans les API

Dans ce chapitre nous nous concentrons sur le concept de sécurité des API, nous présentons d'une part les objectifs de sécurité qui doivent être atteints dans les API et d'autre part nous mettons en évidence les différents niveaux auxquels la sécurité peut être assurée dans une API.

2.1 Les objectifs de sécurité dans les API

Les API Web communiquent en échangeant des messages qui peuvent contenir des données sensibles, il est alors important dans ce cas de protéger ces informations. Il existe en général sept exigences qui doivent être abordées par un outil de sécurité tel que défini par le Norme de sécurité ISO [2] :

- Identification
- Authentification
- Autorisation
- Intégrité
- Confidentialité
- Audit
- Non-répudiation

Dans le cadre des API Web et donc de ce travail nous nous intéressons seulement à certains aspects à savoir l'authentification, l'autorisation, la confidentialité, l'intégrité et la non-répudiation.

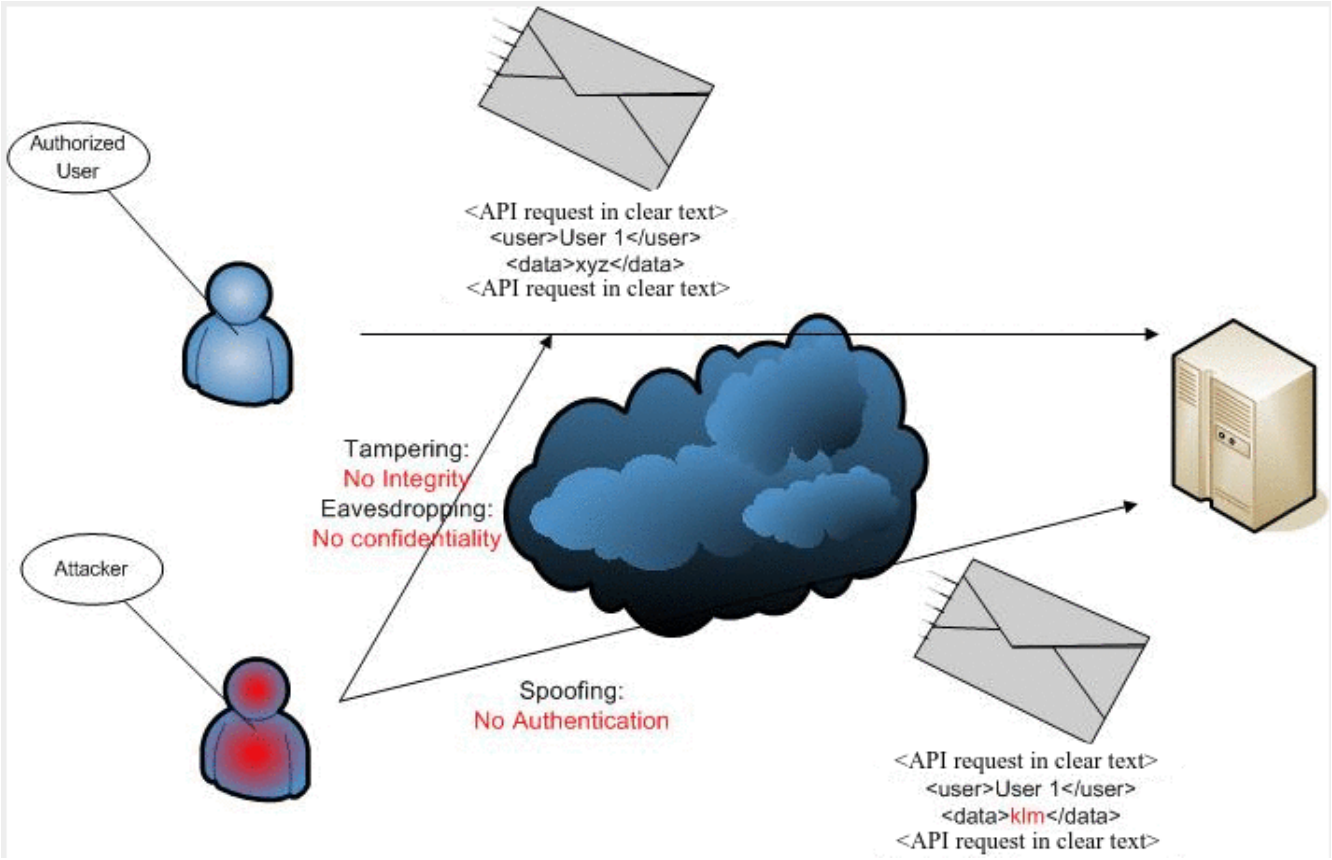

FIGURE 2.1 – Scénario de Web Service sans sécurité (Adaptée de : [2])

La figure ci-dessus présente certaines failles de sécurité auxquelles une API peut être exposée en absence de sécurité. Nous pouvons remarquer par exemple qu'en absence d'authentification, il peut y avoir d'usurpation d'identité(spoofing)

2.1.1 Authentification et Autorisation

L'authentification et l'autorisation sont couramment utilisées ensemble : l'authentification est utilisée pour déterminer de manière fiable l'identité d'un utilisateur final, l'autorisation est utilisée pour déterminer les ressources auxquelles l'utilisateur identifié a accès.

Sur le Web, l'authentification est le plus souvent implémentée via un formulaire qui demande le nom d'utilisateur et le mot de passe. Des clés matérielles et des périphériques externes peuvent être utilisés pour compléter la sécurité des certificats de logiciels. Une fois que l'utilisateur est authentifié, le système décide quelles ressources ou données peuvent être accédées. Pour les API, il est fréquent d'utiliser un type de jeton d'accès, soit obtenu par un processus externe (par exemple lors de la signature de l'API) ou par un mécanisme distinct (par exemple, OAuth). Le jeton est transmis avec chaque demande à une API et est validé par l'API avant de traiter la demande.

2.1.2 Confidentialité, Intégrité, Non-répudiation

La confidentialité de l'information consiste à veiller à ce que l'information soit lue et comprise que par les parties autorisées et auxquelles on a confiance. Toute autre partie sera empêchée d'accéder à l'information, parfois même dans la mesure du possible de ne pas être au courant de son existence. Les données sont souvent classées à différents niveaux de sensibilité, ce qui le rend parfois disponible à tous les utilisateurs autorisés et d'autre fois à un plus petit groupe d'utilisateurs. La confidentialité est souvent assurée par l'utilisation de la cryptographie [22].

L'intégrité de l'information consiste à conserver l'information sous sa forme correcte et à ne pas autoriser sa modification par les parties ou les artefacts sans contrôle. La principale préoccupation de l'intégrité de l'information est de protéger l'information contre les tentatives de manipulation et de s'assurer qu'elle est toujours correct. Dans un système bancaire, nous aimerions que le système détecte si notre information est modifiée par quelqu'un d'autre alors que nous sommes sur le point de procéder à un paiement. Il existe un conflit entre la confidentialité et l'intégrité où la confidentialité concerne la dissimulation de l'information alors que l'intégrité ne se soucie pas si l'information est lue tant qu'elle reste correcte [22].

La non-répudiation est la propriété établissant le lien irréfutable entre un événement et l'entité à son origine, donc qu'un destinataire puisse prouver à tout le monde que l'expéditeur a effectivement envoyé une information. La non-répudiation a pour but de lier différentes parties avec un contrat et les faire signer le contrat avec une signature et un timestamp. C'est une exigence cruciale d'une transaction en ligne, car les signatures numériques associées à un timestamp fournissent à la fois la responsabilité et l'intégrité et, grâce à ces capacités, les parties peuvent s'identifier et garantir que la transaction n'a pas été modifiée depuis le début.

2.2 Les niveaux de sécurité

La sécurité des API Web peut s'effectuer sur deux niveaux d'une part elle peut être assurer en protégeant le canal de communication, dans ce cas on parle de sécurité niveau transport "Transport Level Security", et d'autre part en protégeant les données qui sont échangées lors des communications, on parle de sécurité niveau message "Message Level Secuity".

2.2.1 Transport Level Security

La sécurité au niveau du transport est un mécanisme bien connu et souvent utilisé pour sécuriser les communications HTTP Internet et intranet. La mise en œuvre de la sécurité au niveau du transport revient a assurer la sécurité sur la couche de transport (le réseau) elle-même. La

sécurité au niveau transport, consiste à fournir l'intégrité, la confidentialité et l'authentification du message à mesure qu'il se déplace le long du fil physique. Elle est basée sur l'utilisation du "Secure Sockets Layer" (SSL) ou du "Transport Layer Security" (TLS). Cette approche fournit une sécurité point à point entre les deux points finaux (service et client). S'il existe des systèmes intermédiaires entre le client et le service, chaque point intermédiaire doit transmettre le message sur une nouvelle connexion SSL.

2.2.2 Message Level Security

La sécurité au niveau du message est un service de couche d'application et facilite la protection des données de message entre les applications. Les communications basées sur SOAP présentent la notion de sécurité au niveau du message. Dans la sécurité au niveau du message, les informations de sécurité sont contenues dans le message SOAP, qui permet aux informations de sécurité de voyager avec le message. Par exemple, une partie du message peut être signée par un expéditeur et cryptée pour un destinataire particulier. Lorsque le message est envoyé à partir de l'expéditeur initial, il peut passer par des noeuds intermédiaires avant d'atteindre son destinataire. Dans ce scénario, les parties cryptées continuent d'être opaques à tous les noeuds intermédiaires et ne peuvent être déchiffrées que par le récepteur prévu. Pour cette raison, la sécurité au niveau du message est parfois appelée sécurité de bout en bout.

En résumé nous avons dans ce chapitre montrer d'une part les exigences de sécurité que doivent respecter les API Web et d'autre part les différents niveaux de sécurité auxquels ces exigences peuvent être implémenter.

Chapitre 3

La Sécurité dans les Web Services SOAP

Après avoir présenter les différents aspects de sécurité requis dans les API Web, nous allons dans ce chapitre aborder les différentes méthodes/outils permettant d'assurer ces aspects de sécurité dans les Web Services.

Après avoir présenter les différents aspects de sécurité requis dans les API Web, nous allons dans ce chapitre aborder les différentes méthodes/outils permettant d'assurer ces aspects de sécurité dans les Web Services.

3.1 Security Assertion Markup Language (SAML)

SAML est un mécanisme de sécurité qui permet d'atteindre les objectifs de sécurité telles que l'authentification et l'autorisation.

C'est est un standard de l'OASIS approuvé le 15 mars 2005 [4]. C'est un outil conçu pour l'échange des données liées à la sécurité sous formes d'assertions, entre les autorités SAML (parties). Ces assertions sont définit en XML [4], ce qui rend SAML indépendant de la plate-forme. La création de SAML vient principalement du besoin de fournir l'authentification unique, ce qui fut le cas de la version 1.1. Depuis SAML à évolué et devenu un outil avec plusieurs extensions [10].

La version 2.0 de SAML est la norme leader actuelle pour l'authentification inter-domaine. Elle est supportée par les principaux fournisseurs de services tels que Google, SalesForce, WorkDay, Box, Amazon, et bien d'autres. SAML est une norme pour de vastes réseaux inter-entreprises, gouvernement et éducation autour du globe [10].

Les assertions offrent différents attributs décrivant les sujets. Un sujet peut être une personne ou un périphérique informatique, et les assertions peuvent décrire l'adresse électronique du

sujet, des informations sur les authentifications précédemment effectuées par le sujet, les détails d'autorisation et les décisions quant à savoir si le sujet est autorisé à accéder à certaines ressources [22]. Il existe trois types d'assertions différentes offertes par SAML ; des assertions d'authentification, d'autorisation et d'attribut. Une assertion d'authentification concerne l'authentification d'un sujet et spécifie comment et quand un sujet à été authentifié. Une assertion d'autorisation concerne les règles d'accès et spécifie si un sujet doit avoir accès aux ressources ou non. Une assertion d'attribut peut posséder des éléments d'informations supplémentaires sur un sujet qui peuvent être utiles à une requête [4].

SAML prend en charge les liaisons (bindings) qui constituent le moyen par lequel le fournisseur d'identité redirige l'utilisateur vers le fournisseur de services. Il existe deux types de liens : le HTTP Redirect et HTTP POST binding. La redirection HTTP (HTTP Redirect) utilisera, comme le nom l'indique déjà, une redirection HTTP pour renvoyer l'utilisateur au fournisseur de services. La liaison HTTP Redirect est principalement utile pour les messages SAML courts. Les messages plus longs (par exemple, les messages contenant des revendications SAML signées) doivent être transmis à l'aide d'une liaison HTTP POST [10].

La figure ci-dessous présente le fonctionnement du protocole SAML.

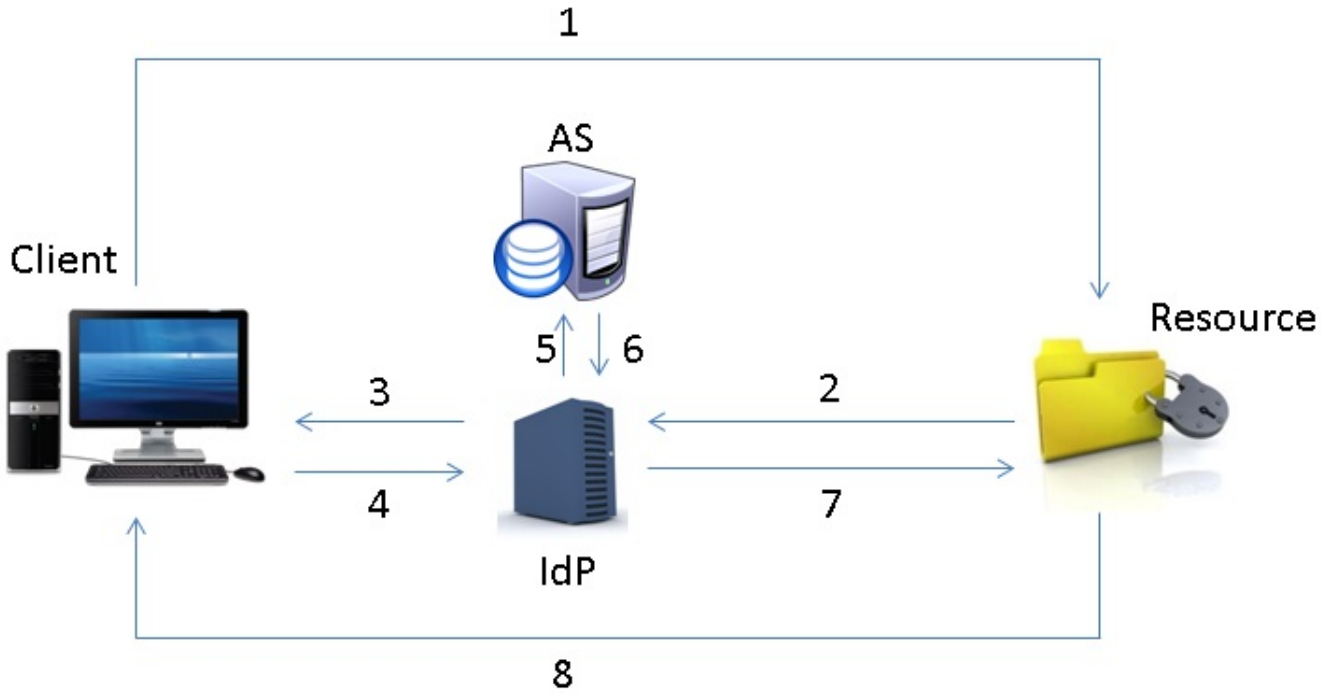

FIGURE 3.1 – Protocole SAML (Source : [10])

Le protocole commence par un client qui contacte le service provider à fin d'accéder à une

ressources (1). le service provider redirige alors le client vers l' IdP et l'AS pour demander l'autorisation d'accéder à la demande (2). Le serveur IdP / AS demande au client de fournir des informations d'identification de connexion (3). Le client renvoie le Les informations d'identification sur le serveur IdP / AS qui, à leur tour, les valide (4). Si les informations d'identification sont valides, l'IdP contacte l'AS et demande des données d'autorisation (5). L'AS renvoie le jeton (token) d'accès à IdP (6), qui informe le service que l'utilisateur est authentifié et envoie un jeton d'accès (7). Le service enregistre le client et accorde l'accès à la ressource demandée (8).

3.2 Web Service Security (WSS)

Le WS-Security est un standard pour sécuriser des messages SOAP en permettant la confidentialité et l'intégrité [22]. Il ne définit pas de nouveaux mécanismes de sécurité mais s'appuie sur les technologies de sécurité existantes telles que la cryptographie et la signature numérique.

WS-Security définit comment sécuriser les messages SOAP en appliquant des technologies de sécurité XML telles que le chiffrement XML et la signature XML [16]. Le chiffrement XML (XML Encryption) définit le processus de cryptage et de décryptage d'un message XML entier, d'une partie d'un message XML ou d'une ressource externe. Comme XML Encryption, la Signature XML (xml Signature) spécifie comment signer numériquement un message XML entier, une partie d'un message XML ou un objet externe. Il définit également comment intégrer différents Jetons (tokens) de sécurité. Les jetons de sécurité fournissent une authentification en prouvant son identité.

Pour sécuriser un message SOAP, le WS-Security étend l'en-tête SOAP en ajoutant un nouvel élément appelé sécurité [16]. L'élément de sécurité gardera donc les références aux éléments cryptés et signés dans l'ensemble du message SOAP. le WS-Security spécifie un attribut qui peut contenir une copie de la signature du message de requête (message) et cette copie fera partie du message de réponse du serveur. Le message de réponse sera d'abord signé par le serveur et envoyé au client De cette manière, la réponse sera liée au message de demande d'origine, une fonction très utile dans le processus de corrélation des messages [22]. Un élément d'horodatage est également spécifié en aidant à prévenir les attaques par rejeu [16].

```
<?xml version="1.0" encoding="utf-8" ?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
                                        xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
  <soap:Header  xmlns:wsse="http://schemas.xmlsoap.org/ws/2002/07/secext"
    xmlns:wsu="http://schemas.xmlsoap.org/ws/2002/07/utility">
    <wsu:Timestamp>
      <wsu:Created  wsu:Id="Id-3beeb885-16a4-4b65-b14c-0cfe6ad26800">2002-08-
       22T00:26:15Z</wsu:Created>
      <wsu:Expires wsu:Id="Id-10c46143-cb53-4a8e-9e83-ef374e40aa54">2002-08-
       22T00:31:15Z</wsu:Expires>
    </wsu:Timestamp>
    <wsse:Security soap:mustUnderstand="1" >
      <xenc:ReferenceList>
        <xenc:DataReference URI="#EncryptedContent-f6f50b24-3458-41d3-aac4-390f476f2e51" />
      </xenc:ReferenceList>
      <xenc:ReferenceList>
        <xenc:DataReference URI="#EncryptedContent-666b184a-a388-46cc-a9e3-06583b9d43b6" />
      </xenc:ReferenceList>
    </wsse:Security>
  </soap:Header>
  <soap:Body>
    <xenc:EncryptedData Id="EncryptedContent-f6f50b24-3458-41d3-aac4-390f476f2e51"
      Type="http://www.w3.org/2001/04/xmlenc#Content">
      <xenc:EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#tripledes-cbc" />
      <KeyInfo xmlns="http://www.w3.org/2000/09/xmldsig#">
        <KeyName>Symmetric Key</KeyName>
      </KeyInfo>
      <xenc:CipherData>
        <xenc:CipherValue>InmSSXQcBV5UiT...  Y7RVZQqnPpZYMg==</xenc:CipherValue>
      </xenc:CipherData>
    </xenc:EncryptedData>
  </soap:Body>
</soap:Envelope>
```

FIGURE 3.2 – Exeampe de message SOAP Sécurisé (Source : [22])

Comme dit plus haut le WS-Security définit comment intégrer des tokens de sécurité dans l'entête SOAP ou dans l'élément sécurité. La spécification définit cinq types (profile) de tokens : Username token profile, X.509 certificate token profile, Kerberos token profile, SAML token profile and, XrML/REL token profile. Il est possible de définir des tokens personnalisés qui ne sont pas dans ce cas standardisés et seront alors incluent comme sous-élément de l'élément sécurité.

3.3 Web Service Policy (WS-Policy)

WS-Policy est l'un des composants clés de l'architecture des services Web, fournissant une grammaire pour décrire les stratégies comme un ensemble d'expressions constituées d'assertions individuelles. Il décrit comment exprimer les exigences requises ou prises en charge par un service Web [24]. Par exemple, il peut indiquer qu'un algorithme de signature spécifique doit être utilisé lors de l'ajout d'une signature numérique. Il sert à placer des exigences générales et n'est pas seulement lié aux exigences de sécurité. Ainsi, pour exprimer des exigences spécifiques à la sécurité, il existe une spécification de politique appelée WS-SecurityPolicy, ce dernier ajoute des assertions relatives à la sécurité au WS-Policy pour lui communiquer des contraintes liées à la sécurité, comme les jetons de sécurité, la confidentialité et l'intégrité [24].

Le WS-Policy offre une manière souple de définir les expressions politiques (stratégiques), laquelle permet à un fournisseur de services Web d'offrir à un consommateur de service le choix de remplir tout ou une partie de la politique. Différents choix peuvent être définis pour que le client puisse fournir des informations dont il possède ou dont il a connaissance, ce qui permet d'économiser le temps et d'éviter au client de recueillir de nouvelles informations. En outre, les politiques peuvent être spécifiées sous deux formes, forme compacte et normale où la dernière est la plus détaillée.

```
<Policy>
 <All>
  <wsap:UsingAddressing />
  <ExactlyOne>
   <sp:TransportBinding>...</sp:TransportBinding>
   <sp:AsymmetricBinding>...</sp:AsymmetricBinding >
  </ExactlyOne>
 </All>
</Policy>
```

FIGURE 3.3 – Scénario de Web Service sans sécurité (Source : [22])

```
<Policy>
 <ExactlyOne>
  <All>
   <wsap:UsingAddressing />
   <sp:TransportBinding>...</sp:TransportBinding>
  </All>
  <All>
   <wsap:UsingAddressing />
   <sp:AsymmetricBinding>...</sp:AsymmetricBinding >
  </All>
</ExactlyOne>
</Policy>
```

FIGURE 3.4 – Scénario de Web Service sans sécurité (Source : [22])

Les deux figures précédentes montrent la même politique spécifiée dans les deux formes. Nous voyons que le fournisseur de services exige que WS-Addressing fasse partie du message SOAP et que les échanges SOAP suivants doivent être sécurisés soit par la sécurité au niveau du transport, soit par la sécurité au niveau du message [22]. Dans La première figure, il est exigé que toutes les expressions de politiques soient remplies et dans la deuxième le client peut faire le choix de respecter une seule parmi celles proposées.

3.4 Web Service Trust

WS-Trust est une extension de WS-Security qui fournit des méthodes pour demander, émettre, renouveler, annuler et valider des jetons de sécurité. En outre, il définit les moyens d'évaluer la présence de relation de confiance entre partis ou d'en établir [25]. Il utilise le WS-Security pour transférer les jetons de sécurité requis, en utilisant la Signature XML et l'Encryption XML pour assurer la confidentialité. Il peut utiliser le WS-Policy pour spécifier les tokens de sécurité requis par la cible.

La partie centrale de ce standard est le service de token de sécurité (STS) qui est un service Web fournissant à un client des méthodes qui émettent, renouvelle, annule ou valide différents types de jetons. Le STS peut également être considéré comme le courtier d'authentification fournissant une infrastructure de contrôle d'accès commune et est responsable de la négociation de la confiance entre un client et un service Web [22]. Le STS est clairement utile dans une situation dans laquelle il n'y a pas de confiance directe entre un service et son consommateur, mais tous deux font confiance au STS d'une façon ou d'une autre.

Dans un scénario typique de WS-Trust, seul le client peut initier une communication avec le STS, il devient donc de la responsabilité du client de récupérer le jeton (token) de sécurité requis par le web service auprès du STS et l'envoyer au service Web d'entreprise. Le client prend connaissance du token requis en consultant la politique du web service définit grâce au WS-Policy et au WS-SecurityPolicy. Pour communiquer avec le STS, le client doit ajouter un élément supplémentaire appelé "RequestSecurityToken" (RTS) dans sa requête SOAP. l'élément RTS contiendra toutes les informations que le STS doit comprendre, comme par exemple émettre un nouveau jeton Kerberos. Lorsqu'un jeton est prêt à être livré au client, le STS crée une réponse SOAP et l'étend avec un "RequestSecurityTokenResponse" (RSTR). L'élément RSTR héberge ou fournit une référence à l'élément jeton , et peut également fournir une clé de session afin que toute la communication ultérieure entre le client et le service soit cryptée à l'aide de cette clé. La clé de session est une clé temporaire et dure aussi longtemps que la session entre les deux parties est active.

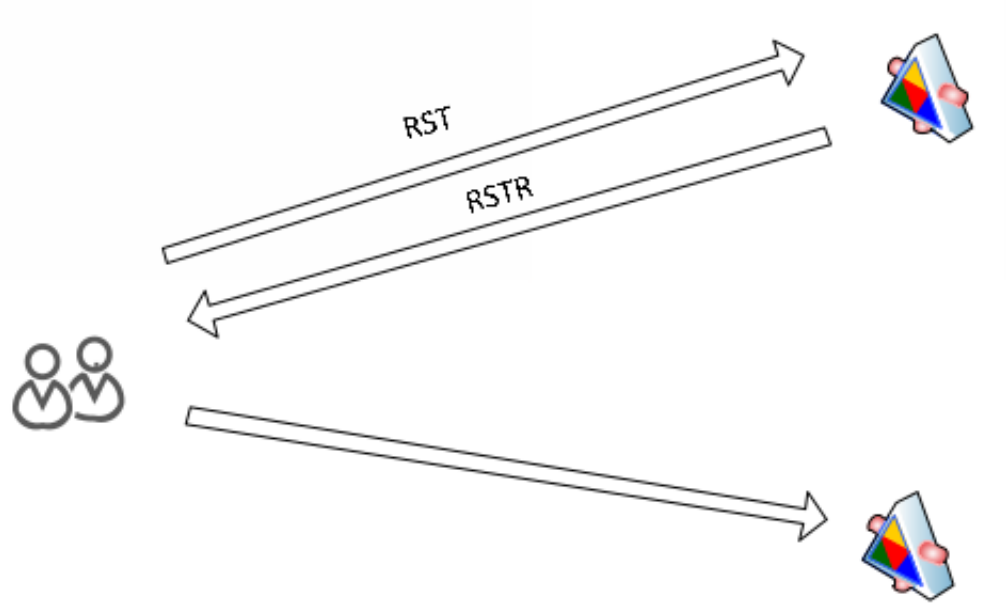

FIGURE 3.5 – Déroulement du scénario WS-Trust (Source : [22])

3.5 Web Service Federation

Une fédération est une collection de domaines (domaines de sécurité) qui ont établi des relations pour partager des ressources en toute sécurité. Un fournisseur de ressources dans un domaine peut fournir un accès autorisé à une ressource qu'il gère en fonction des réclamations (claims) concernant un principal (comme l'identité ou d'autres attributs distinctifs) qui sont prouvées par un fournisseur d'identité (ou tout service de jeton de sécurité) dans un autre domaine.

L'objectif de la fédération est de permettre la sécurité des identités principales et des attributs à partager entre les limites de confiance selon les politiques établies. Les politiques dictent, entre autres, les formats et les options, ainsi que les fiducies et les exigences de confidentialité et de partage [26]. Il existe trois type de gestion d'identité ; la gestion fédéré isolé, centralisé et distribué [22].

Le WS-Federation est de la catégorie distribué, c'est un standard de L'OASIS [25]. Conçu au dessus des standards tels que le WS-Security, le WS-SecurityPolicy, le WS-Policy, le WS-Trust, il fournit une architecture d'identité fédérée pour l'architecture des services Web [22]. Le WS-Federation définit des mécanismes permettant à différents domaines de sécurité de se fédérer.

Il décrit comment les scénarios de confiance fédérés peuvent être construits en utilisant WS-Security, WS-Policy, WS-Trust et WS-SecureConversation.

Le déroulement d'un processus d'identité avec le WS-Federation est basé sur l'utilisation du WS-Trust. Pour que le client puissent utiliser un service d'un autre domaine, il doit d'abord demander un token de sécurité auprès de son fournisseur d'identité (IdP) qui n'est qu'un STS avec des extensions. L'IdP dans le domaine du client est un tiers de confiance pour le client et le service, et est responsable de fournir au client toute type de token dont il à besoin. Pour passer du token du domaine du client à celui requis par le service, un service de pseudonyme est nécessaire. Ce dernier fournit des mécanismes pour enregistrer et obtenir d'autres informations sur une identité, offrir un changement d'identité ou de token dans un scénario de domaine interdisciplinaire, par exemple en changeant un certificat X.509 pour un jeton Kerberos [22].

3.6 Web Service SecureConversation

Le WS-Security est utilisé pour assurer la confidentialité et l'intégrité d'un seul message SOAP, son modèle devient trop coûteux s'il est nécessaire d'effectuer plusieurs échanges de demandes / réponses entre un client et le serveur. Le WS-SecureConversation introduit la notion de jeton de contexte de sécurité composé d'un secret partagé entre les deux parties, utilisé pour faire respecter la confidentialité et l'intégrité lors d'échanges de messages multiples au cours d'une période de session donnée [23]. Cette spécification utilise WS-Security, WS-Trust et WS-Policy pour négocier et émettre des clés de session.

Le token de sécurité de contexte doit être cré et échanger entre les parties avant qu'une session ne débute. cette échange peut s'effectuer de 3 façons : soit, un participant crée le token et le transmet à l'autre, soit faire la demande auprès d'un STS qui le distribue aux deux parties ou encore utilisé le protocole de négociation en 4 étapes du WS-Trust [22].

L'échange et l'authentification complète des clés ne sont nécessaires que lors de l'établissement d'un jeton de contexte de sécurité. Après cette phase, chaque message est crypté ou signé avec les informations contenues dans le jeton du contexte de sécurité. Le secret qui fait partie du jeton du contexte de sécurité peut être utilisé pour crypter et signer les messages, mais la norme recommande l'utilisation de clés dérivées créées à partir du jeton de contexte secret. On peut aussi créer plusieurs Clés dérivées selon les exigences. Par exemple, nous pouvons utiliser une clé pour signer et d'autres pour chiffrer le message SOAP. Étant donné que tous les messages SOAP appartenant au même échange de message feront référence au même jeton de contexte de sécurité, on augmentera la performance globale [23].

3.7 Relations entre les standards de sécurité des services Web SOAP

La figure ci-dessous illustre la relation entre ces normes de sécurité des services Web SOAP.

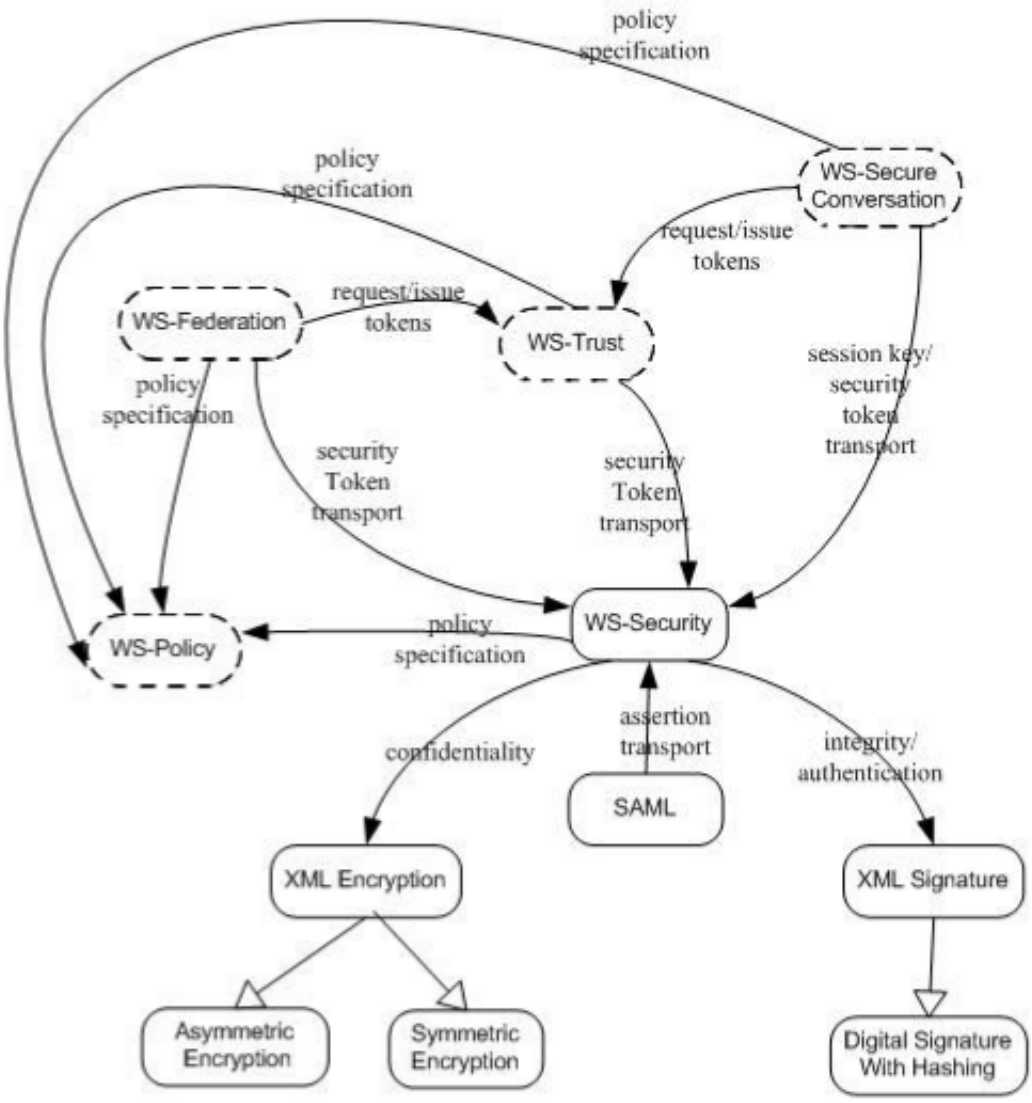

FIGURE 3.6 – (Source : [9])

Nous pouvons remarquer que le WS-Security utilise le XML Encryption, le XML Signature respectivement la confidentialité et l'intégrité, ce dernier tout comme le WS-Federation, le WS-Trust, et le WS-SecureConversation utilisent le WS-Policy pour assurer et spécifier les politiques de sécurité. Le WS-Security est utilisé par SAML,WS-Trust, WS-Federation et WS-

SecureConversation pour assurer la sécurité des assertions SAML, et des tokens de securité. Le WS-Federation et le WS-SecureConversation utilisent le WT-Trust pour les requêtes de tokens.

Dans ce chapitre nous avons mis en évidences les standards de sécurité définis pour assurer la sécurité des Web API SOAP et montrer les relations entre ces dernières.

Chapitre 4

La sécurité dans les API Restful

Dans ce chapitre nous allons aborder la sécurité au niveau des API Web Restful, d'une part nous présenterons les efforts entrepris et d'autres part nous décrirons quelques solutions libres utilisées fréquemment.

Dans ce chapitre nous allons aborder la sécurité au niveau des API Web Restful, d'une part nous présenterons les efforts entrepris et d'autres part nous décrirons quelques solutions libres utilisées fréquemment.

4.1 Quelques efforts message level secuity

Contrairement aux Web services SOAP, il n'existe pas de standard de sécurité définit pour assurer la sécurité au niveau message au niveau des API Restful, . Néanmoins des effort ont été entrepris par des communautés afin de répondre à ce besoin.

Comme l'explique Omar Slomic [22], Dan R. Olsen de l'Université de Brigham Young tente dans sa thèse de répondre à ce besoin. La thèse traite de la façon dont les éléments de la spécification WS-Security peuvent être inclus dans l'en-tête HTTP. Ceci est démontré par deux exemples différents où le premier montre comment un URI peut être signé et comment cette signature de l'URI peut être stocké dans l'en-tête sous forme d'élément WS-Security. Un autre exemple montre comment l'élément UsernameToken de WS-Security peut être stocké dans l'en-tête. Cela signifie que la thèse ne traite pas de la protection du message dans le corps HTTP.

Aussi Dan Forsberg, chercheur au Nokia Research Centre à l'époque aborde ce problème dans son article . L'article délibère sur le cryptage de toutes sortes de contenu HTTP comme les images, les fichiers, etc. Le même auteur se concentre fortement sur la mise en cache HTTP et traite d'une façon de fournir des clés de décryptage sur les sessions TLS. mais cet article ne met pas l'accent sur les signatures ou le cryptage partiel du contenu [22].

Omar Slomic aborde aussi le problème dans sa thèse de master dans laquelle il propose un prototype de solution pour la sécurité au niveau message pour les services RESTful. La solution est pour la plupart technique et utilise des mécanismes multiplate-forme qui sont composés ensemble, tandis qu'une partie plus petite de la solution traite d'une approche non technique concernant la distribution des token [22].

Comme effort pour satisfaire au besoin de la sécurité niveau message pour les API Restful nous pouvons citer aussi, Gabriel Serme et & al qui présentent dans leur article un protocole de sécurité REST pour fournir une communication sécurisée, ainsi qu'une analyse de performance de celle-ci par rapport à la configuration équivalente du WS-Security. Ce protocole fournit le chiffrement, la signature et les deux à la fois grâce à des tags incluent dans l'entête HTTP [21].

4.2 Quelques Outils libres de sécurité

Bien que des standards de sécurité n'ont pas été définit par l'approches REST, il existe plusieurs outils libre qui émergent dans le but d'offrir certains aspects de sécurité. Dans cette section nous présentons les plus connus et fréquemment utilisés.

4.2.1 Transport Layer Security (TLS)/ Secure Socket Layer (SSL)

C'est un outil de sécurité qui permet d'atteindre les objectifs de sécurité que sont l'authentification, la confidentialité et l'intégrité.

Le TLS/SSL est un standard IETF (Internet Engineering Task Force) pour sécuriser un canal de communication entre un client et un serveur [6]. Un échange TLS commence par une série d'échange de messages dans lesquels le serveur (et éventuellement le client) est authentifié à l'aide de certificats X.509 et une clé de session symétrique est établie [6]. TLS fournit un cryptage de couche de transport, une protection d'intégrité et une authentification. De nombreux protocoles d'application peuvent être envoyés sur un canal TLS-sécurisé, y compris HTTP sous la forme de HTTP Secure (HTTPS). TLS 1.2 est la version la plus récente du protocole, mais l'utilisation de celui-ci n'est pas encore omniprésente, de nombreux progiciels ne supportant que les versions TLS 1.1 et ci-dessous [19].

Le protocole Handshake est responsable de l'authentification du client et du serveur, et d'échanger des clés cryptographiques et de négocier un algorithme de cryptage entre eux [6]. L'information se trouve généralement en texte clair dans les deux points d'extrémité, mais est protégée par le tunnel chiffré que l'information traverse . Ce type de sécurité est appelé sécurité point à point puisque les informations sont sécurisées entre les deux nœuds.

La communication entre le client et le serveur débute par une requête du client pour la mise en place de connexion sécurisée par TLS, ainsi le client peut décider des algorithmes cryptographiques qu'il veut utiliser. Après cette étape de négociation, le serveur envoie son certificat au client, celui contient sa clé publique, ses informations ainsi qu'une signature numérique sous forme de texte chiffré appartenant à son autorité de certificat (CA). Après avoir reçu le certificat, le client utilise la clé publique de CA pour identifier la signature du certificat. Ensuite, le client génère une clé de session et la crypte à l'aide de la clé publique du serveur. La clé de session cryptée est alors envoyée au serveur qui la déchiffre en utilisant sa propre clé privée. Le client et le serveur commencent à s'échanger des données en chiffrant celles-ci avec la clé de session qu'ils ont en commun.

4.2.2 OAuth 2.0

OAuth 2.0 est un outil de sécurité qui a pour objectif d'atteindre l'objectif de sécurité qu'est l'autorisation.

C'est est une norme de l'IETF [19] approuvée en octobre 2012. C'est un framework d'autorisation «basé sur la délégation d'accès». Ce dernier permet à une application tierce (Client) d'obtenir un accès limité à un service HTTP, soit en tant que propriétaire de la ressource (Resource Owner) en organisant une interaction d'approbation entre le propriétaire de la ressource et le service HTTP, soit en permettant à l'application tierce d'obtenir un accès personnel [8]. La version actuelle de OAUTH est la 2.0 et elle est incompatible avec ça précédente version. OAuth 2.0 définit quatre rôles représentant les différentes entités intervenant dans l'exécution du protocole [8] :

- **Resource Owner (RO)** :
  Une entité capable d'autoriser l'accès à une ressource protégée, Lorsque le propriétaire de la ressource est une personne, il est appelé un utilisateur final.
- **Resource Server** :
  Le serveur hébergeant les ressources protégées, capable d'accepter et répondre aux demandes de ressources protégées en utilisant des jetons d'accès.
- **Client** :
  Une application, un site web ou tout système sollicitant l'accès à une ressource protégé au nom (de la part) du Resource Owner et de son autorisation.
- **Authorization Server (AS)** :
  Le serveur fournissant le jeton d'accès au client, après avoir authentifié avec succès le Resource Owner et obtenu son autorisation.

L'idée est de créer un code d'autorisation anonyme générer par l'utilisateur final (Resource owner) Via le fournisseur d'identité (Authorization Server) qui joue le rôle d'un tiers de confiance

afin d'autoriser l'application consommatrice (Client) à accéder aux ressources contrôlées par le fournisseur de services (Resource Server). Le client utilise par la suite ce code d'autorisation pour obtenir un jeton d'accès qui lui permet d'accéder aux ressources autorisées par le Resource owner.

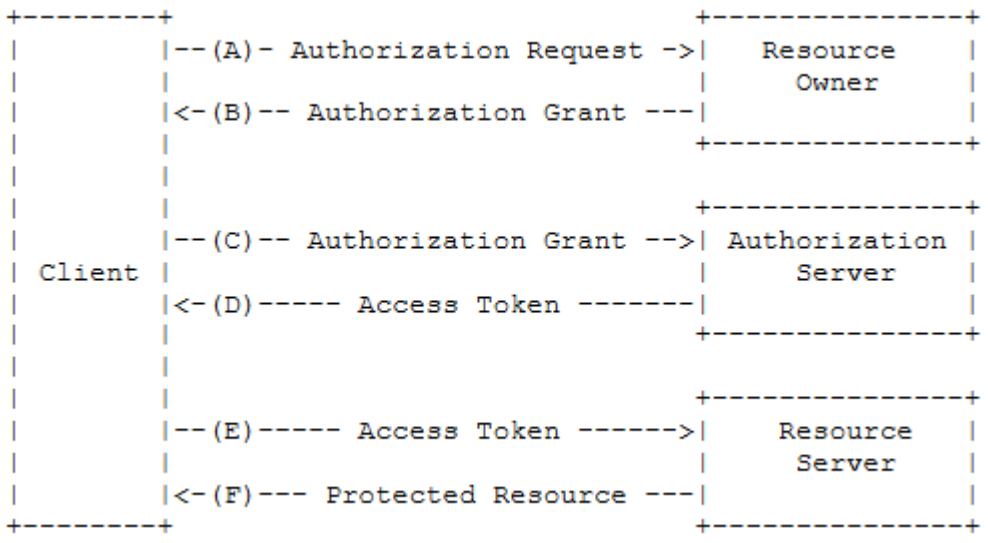

FIGURE 4.1 – OAuth Abstract Protocol Flow (Source : [8])

Le protocole se déroule en trois principales phases : l'enregistrement du client, l'obtention de l'autorisation et l'accès du client aux ressources.

Enregistrement du Client (Client Registration)

Selon OAuth 2.0 avant d'initier le processus d'autorisation le client devrait s'enregistrer auprès du serveur d'autorisation. Cela n'exclue pas l'utilisation des clients non enregistrés qui nécessitera des analyses de sécurité supplémentaires [8]. Cette phase d'enregistrement consiste en un échange d'informations du client entre le serveur d'autorisation et ce dernier.

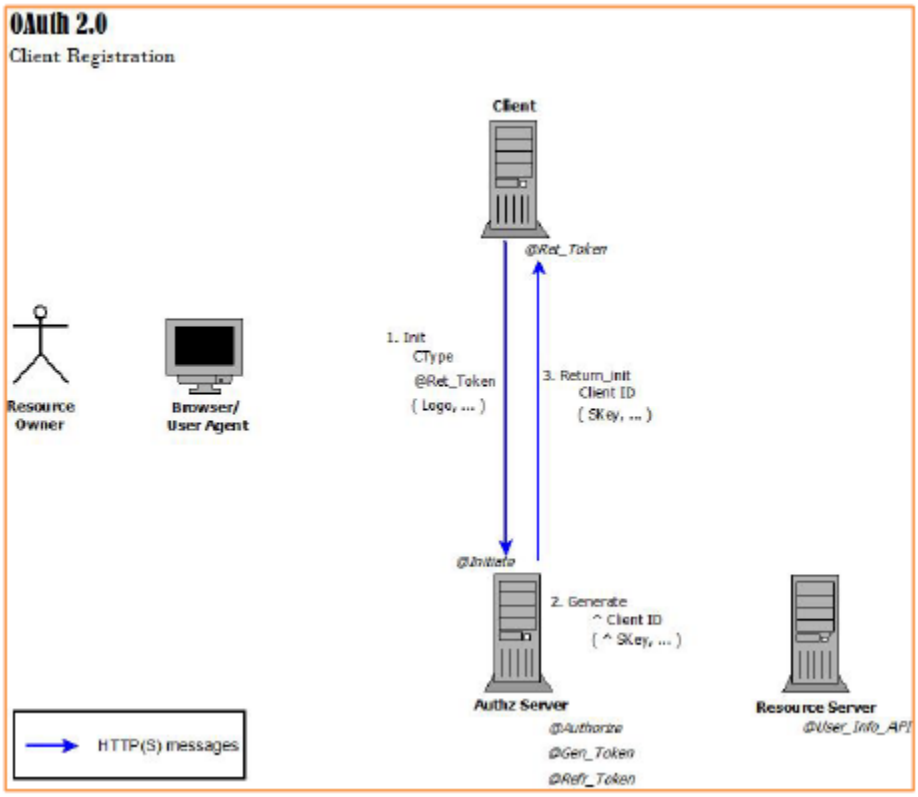

FIGURE 4.2 – OAuth Client Registration (Source : [15])

Ainsi à travers le point d'initialisation offert par le serveur d'autorisation, le client fournis des informations incluant son type (public ou confidentiel) et le point de redirection vers lequel le serveur d'autorisation devra envoyé les informations d'autorisation. Le client peut aussi échanger d'autres informations telles que : son nom, sa description, son logo etc ...

Le serveur d'autorisation génère ensuite un identifiant unique "Client ID" qu'il renvoi au client. Le serveur peut aussi renvoyé une clé secret qui peut être utiliser pour des échanges d'informations confidentielles.

L'obtention de l'autorisation

Pour obtenir le token d'accès , le client doit obtenir l'autorisation du propriétaire de la ressource. Pour cela OAuth 2.0 définit quatre principaux types d'autorisation pour les applications clientes à savoir : l'accès par code d'autorisation (authorization code), l'accès implicit(implicit), l'accès basé sur l'utilisation du mot de passe du Resource Owner (resource owner password credentials), et l'accès basé sur les informations d'identité du client (client credentials). Il offre aussi un mécanisme d'extensibilité pour définir des types supplémentaires [8]. Nous présentons dans ce rapport que les 2 types les plus fréquemment utilisés à savoir l'accès par code

d'autorisation (authorization code grant),et l'accès implicit(implicit grant). Des figures présentant les deux autres son fournies en annexe.

- **Authorization Code** :

  Ce type d'autorisation est utilisé pour obtenir à la fois les jetons d'accès (access token) et les jetons de mise à jour (refresh token) et est optimisé pour les clients confidentiels, c'est à dire les applications clientes qui sont en mesure de maintenir la confidentialité de leurs titres de compétences [15].

  Le code d'autorisation est obtenu en utilisant un serveur d'autorisation en tant qu'intermédiaire entre le client et le propriétaire de la ressource. Au lieu de demander l'autorisation directement au propriétaire de la ressource, le client dirige le propriétaire de la ressource vers un serveur d'autorisation (via son user-agent) qui, à son tour, redirige le Le propriétaire de la ressource vers au client avec le code d'autorisation.

  Avant de diriger le propriétaire de la ressource vers le client avec le code d'autorisation, le serveur d'autorisation authentifie le propriétaire de la ressource et obtient l'autorisation. Étant donné que le propriétaire de la ressource ne s'authentifie que avec le serveur d'autorisation, les informations d'identification du propriétaire de la ressource ne sont jamais partagées avec le client.

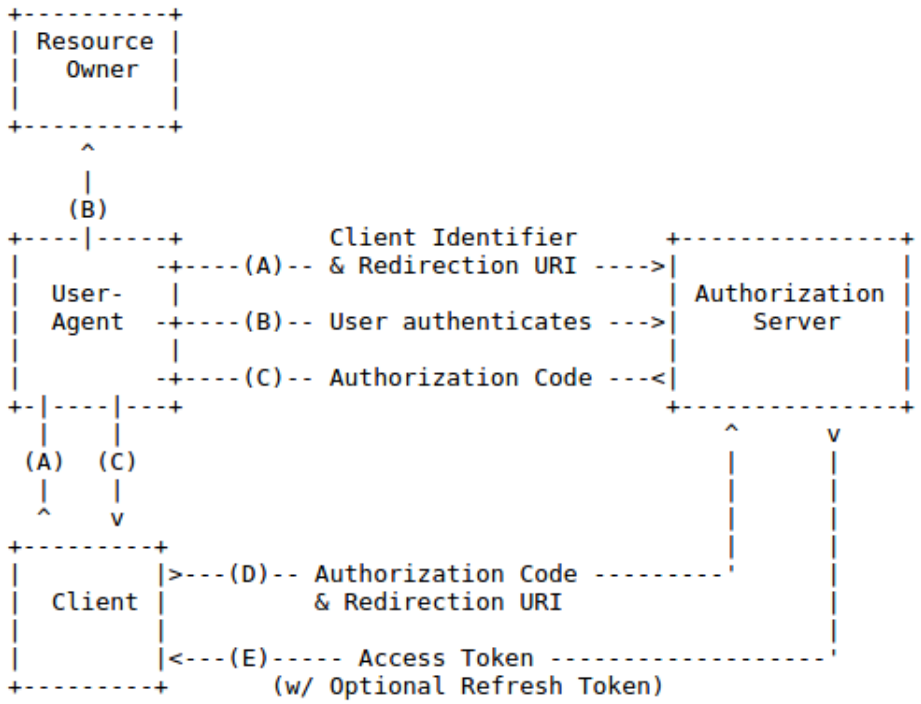

FIGURE 4.3 – OAuth Authorization Code Flow (Source : [8])

- **Implicit** :

L'implicit grant est un code d'autorisation simplifié conçu pour les applications clientes qui ne peuvent garantir la confidentialité de leurs informations d'identification, telles que les applications implémentées dans un navigateur à l'aide d'un langage de script tel que JavaScript [8] ou les applications natives exécutées sur le périphérique utilisé par le propriétaire de la ressource [15].

Dans le flux implicite, au lieu de délivrer au client un code d'autorisation, le client reçoit directement un jeton d'accès (en raison de l'autorisation du propriétaire de la ressource). Le type d'autorisation est implicite, car aucune donnée intermédiaire (comme un code d'autorisation) n'est émise (et plus tard utilisée pour obtenir un jeton d'accès).

Lors de l'émission d'un jeton d'accès pendant le flux d'autorisation implicite, le serveur d'autorisation n'authentifie pas le client. Dans certains cas, l'identité du client peut être vérifiée via l'URI de redirection utilisé pour fournir le jeton d'accès au client. Le jeton d'accès peut être exposé au propriétaire de la ressource ou à d'autres applications avec l'accès à l'agent utilisateur du propriétaire de la ressource.

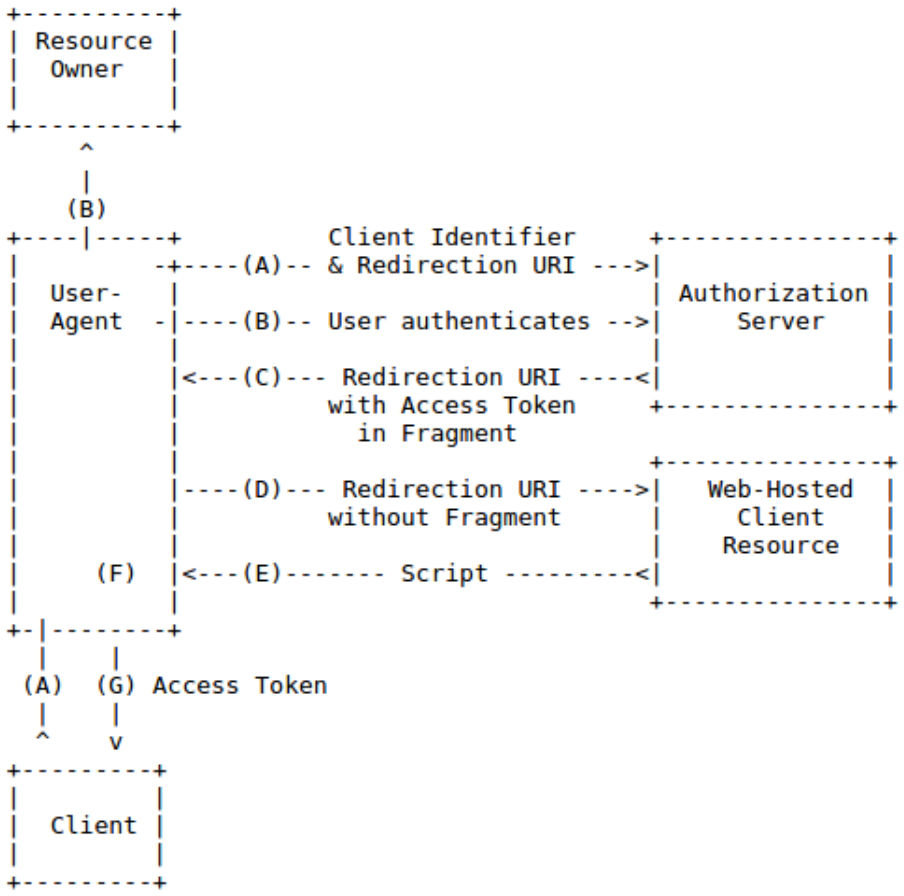

FIGURE 4.4 – Auth Implict Flow (Source : [8])

4.2.3 OpenID Connect

OpenID connect s'inscrit dans la catégorie des méthodes de sécurité permettant d'atteindre les objectifs de sécurité que sont l'authentification et l'autorisation.

C'est une couche d'identité simple au-dessus du protocole OAuth 2.0. Il permet aux clients de vérifier l'identité de l'utilisateur final en fonction de l'authentification effectuée par un serveur d'autorisation, ainsi que d'obtenir des informations de profil de base sur l'utilisateur final de manière interopérable et en mode REST [20]. En effet, OpenID Connect 1.0 est fondamentalement une redéfinition d'OAuth 2.0 résultant de la fusion et de l'amélioration de OpenID et OAuth 2.0 en fournissant des informations d'authentification de l'utilisateur tout en essayant de surmonter les principaux problèmes détectés avec l'adoption d'OAuth 2.0 [15].

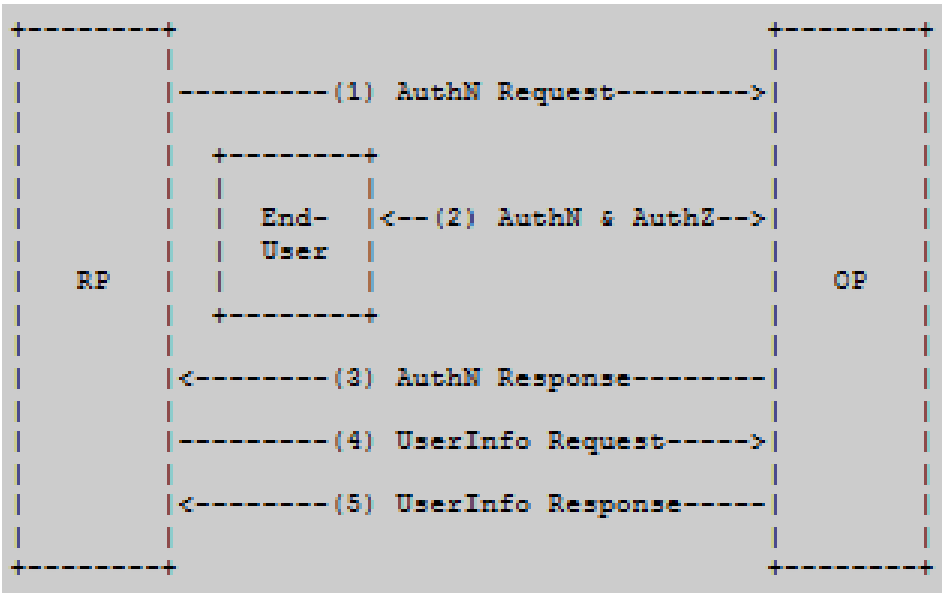

FIGURE 4.5 – OpenID Connect Abstract Protocol Flow (Source : [20])

Étant basé sur le framework oauth 2.0, OpendID Connect conserve essentiellement la même structure que que ce dernier et se focalise sur la sécurité du protocole et l'authentification de l'utilisateur final. Ainsi, il peut également se diviser en trois parties principales : l'enregistrement du client, l'autorisation du propriétaire de la ressources et l'accès des clients. Le protocole définit trois flows possibles à savoir : l'authorization code, l'implicit qu'il hérite de oauth 2.0 auxquels il ajoute le flow hybride qui est la combinaison des deux autres.

Le tableau ci-dessus présente le caractéristiques de ces trois flows.

TABLE 4.1 – Les flows d'authentification OpenID Connect (Source : [20])

Propriété	**Authz Code Flow**	**Implicit Flow**	**Hybrid Flow**
Toutes les jetons sont retournés du Authz Endpoint	Non	Oui	Non
Toutes les jetons sont retournés du Token Endpoint	Oui	Non	Non
Les Tokens ne sont pas révélés au User-Agent	Oui	Non	Non
Le client peut être authentifié Possibilité de mettre à jour les tokens	Oui Oui	Non Non	Oui Oui
Communication en aller-retour	Non	Oui	Non
Communication serveur à serveur	Oui	Non	Varie

Enregistrement du Client (Client Registration)

Pour qu'un client (RP) OpenID Connect puisse utiliser des services OpenID Connect d'un utilisateur final, ce dernier doit s'inscrire auprès du fournisseur OpenID (OP) afin de lui fournir des informations le concernant, et d'obtenir les informations nécessaires à son utilisation, y compris un identifiant client OAuth 2.0 [20],. La spécification exige que l'enregistrement du client soit toujours effectué de manière dynamique via le point final d'inscription "Initiate" publié par le fournisseur OpenID [15].

La figure ci-dessous présente le déroulement du processus d'inscription d'un client auprès du fournisseur OpenID.

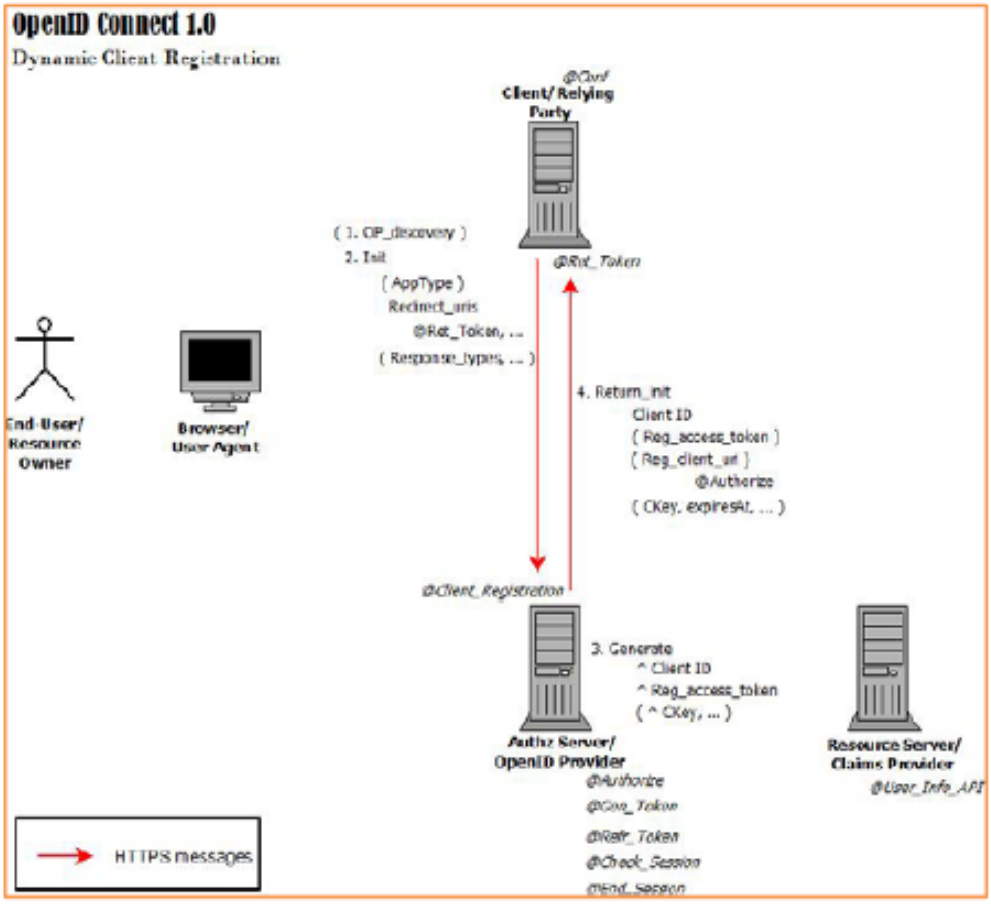

FIGURE 4.6 – Enrégistrement dynamique du client OpenID Connec (Source : [15])

Le client utilise le protocole de découverte ou tout autre mécanisme pour localiser le point d'initiation (endpoint initiate) du OPenID Provider. Après l'obtention de cette information, le client envoie une requête HTTPS Post avec tous les paramètres pour s'inscrire sous la forme d'un objet JSON, la valeur du paramètre "response_type" détermine le type de choisi par le

client. Le fournisseur OpenID génère alors un nouveau clientID, le jeton d'accès à l'enregistrement à utiliser par le client lors de la demande d'opérations supplémentaires et éventuellement d'une clé secrète pour sécuriser les communications entre le client et le fournisseur Open-dID

Authentification de l'utilisateur

Comme dit plus haut OPenID Connect définit trois flow pour effectuer l'authentification :

- **Authorization code** :

  Le flux de code d'autorisation renvoie un code d'autorisation au client, qui peut ensuite l'échanger pour un jeton d'identification et un jeton d'accès directement. Cela offre l'avantage de ne pas exposer de jetons à l'user-agent et éventuellement à d'autres applications malveillantes ayant l'accès au user-agent. Le serveur d'autorisation peut également authentifier le client avant d'échanger le code d'autorisation pour un jeton d'accès. Le flux du code d'autorisation est adapté aux clients qui peuvent assurer en toute sécurité un secret client entre eux et le serveur d'autorisation [20]

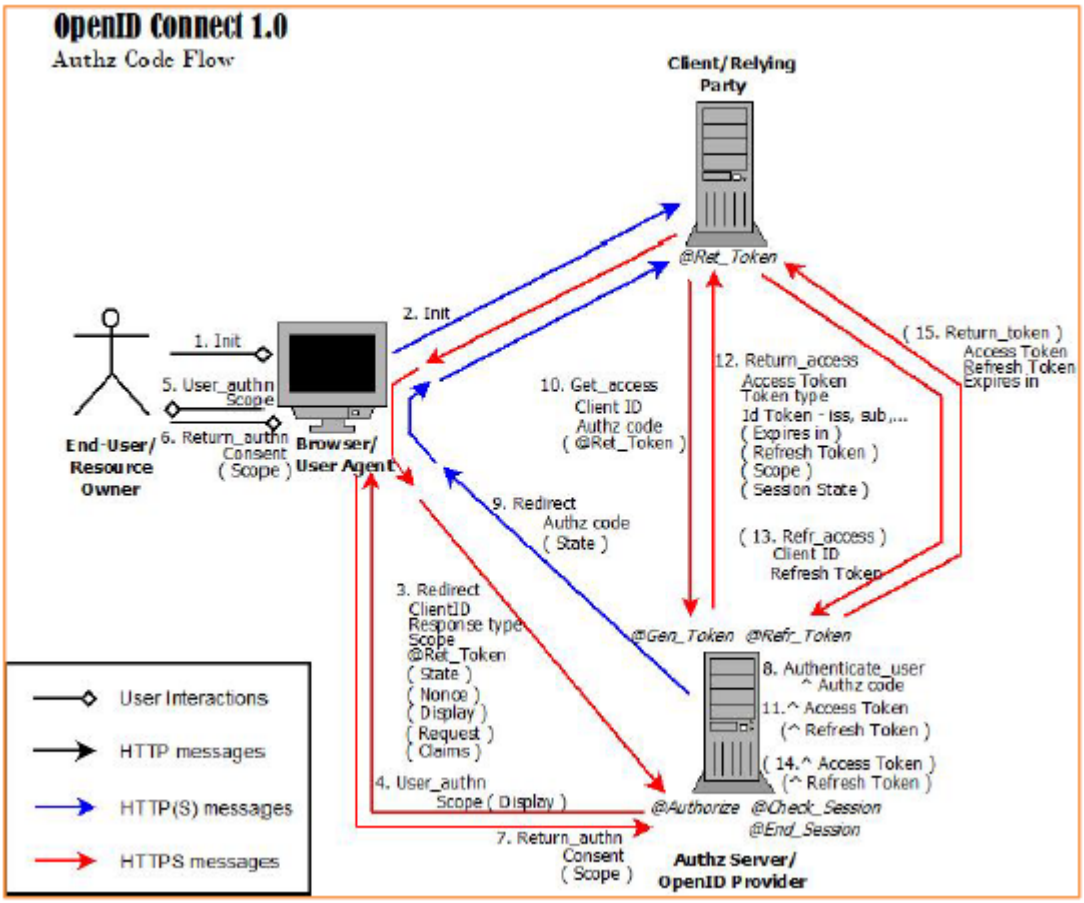

FIGURE 4.7 – OpenID Connect Code Authorization flow (Source : [15])

L'utilisateur final initie une requête vers le client pour accéder à ses services. Le client prépare alors une demande d'authentification contenant les paramètres de demande souhaités. Ensuite ce dernier envoie la requête au serveur d'autorisation. Le Serveur d'autorisation authentifie alors l'utilisateur final puis obtient son consentement. Le serveur d'autorisation renvoie ensuite l'utilisateur final au client avec un code d'autorisation. Le client fait ensuite une requête au "Token Endpoint" du serveur d'autorisation, il reçoit alors une réponse contenant un tokenID et un token d'accès. Finalement le client valide le token ID et récupère l'identificateur de sujet de l'utilisateur final.

- **Implicit flow** :

The Implicit Flow est principalement utilisé par les clients implémentés dans un navigateur à l'aide d'un langage de script. Le jeton d'accès et le jeton d'identification sont retournés directement au client, ce qui peut les exposer à l'utilisateur final et aux appli-

cations qui ont accès à l'agent utilisateur de l'utilisateur final. Le serveur d'autorisation n'effectue pas l'authentification client [20].

La figure décrit le déroulement de l'implicit flow.

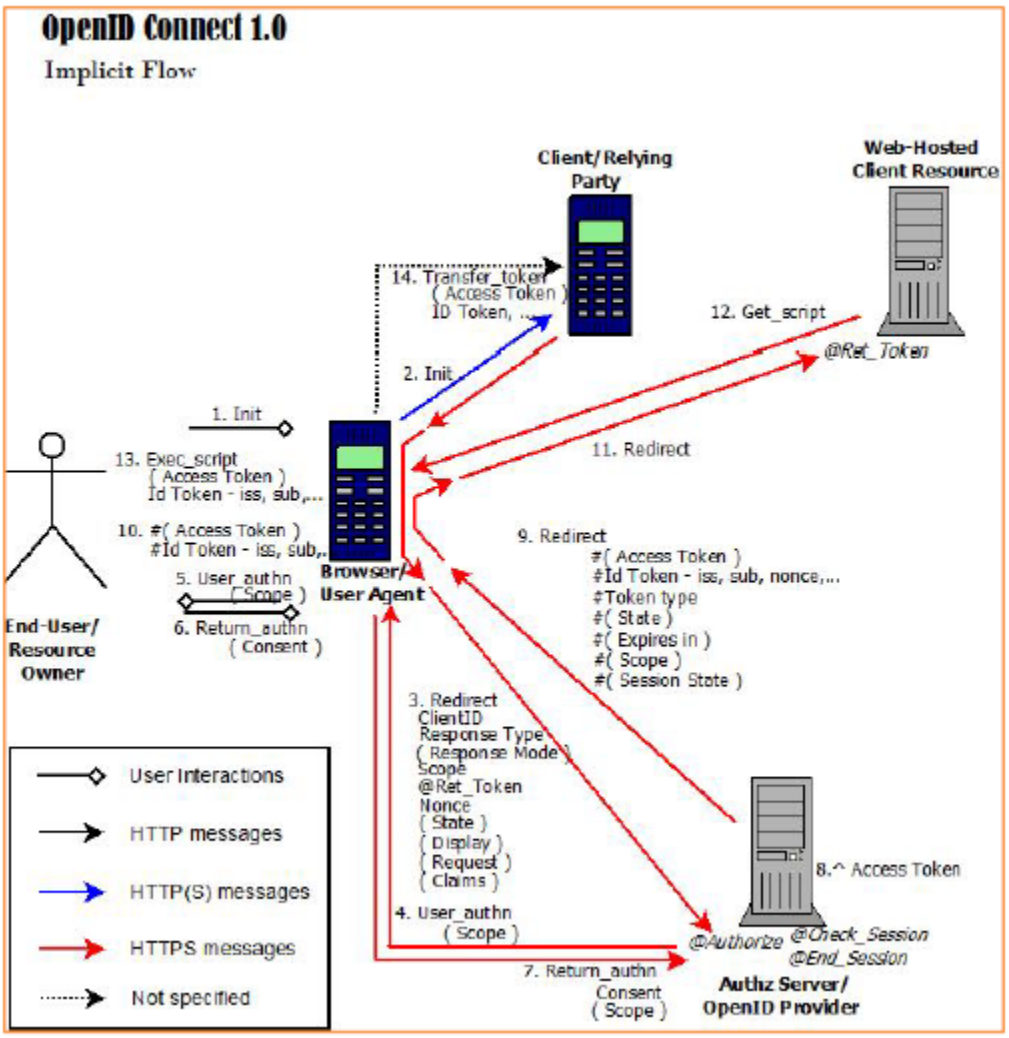

FIGURE 4.8 – OpenID Connect Implicit flow (Source : [15])

Ce protocole est très similaire au implicit flow du framework OAuth 2.0 mais présente les principales fonctionnalités décrites spécifiquement pour l'implicit flow de OpenID Connect [15].

- **Hybrid flow** :

Le flow hybride est la combinaison des deux précédents flows, ainsi il combine les certaines caractéristiques des deux. Lors de l'utilisation du flux hybride, certains jetons sont retournés à partir du "Authz Endpoint" et d'autres sont retournés à partir du "Token Endpoint" [20].

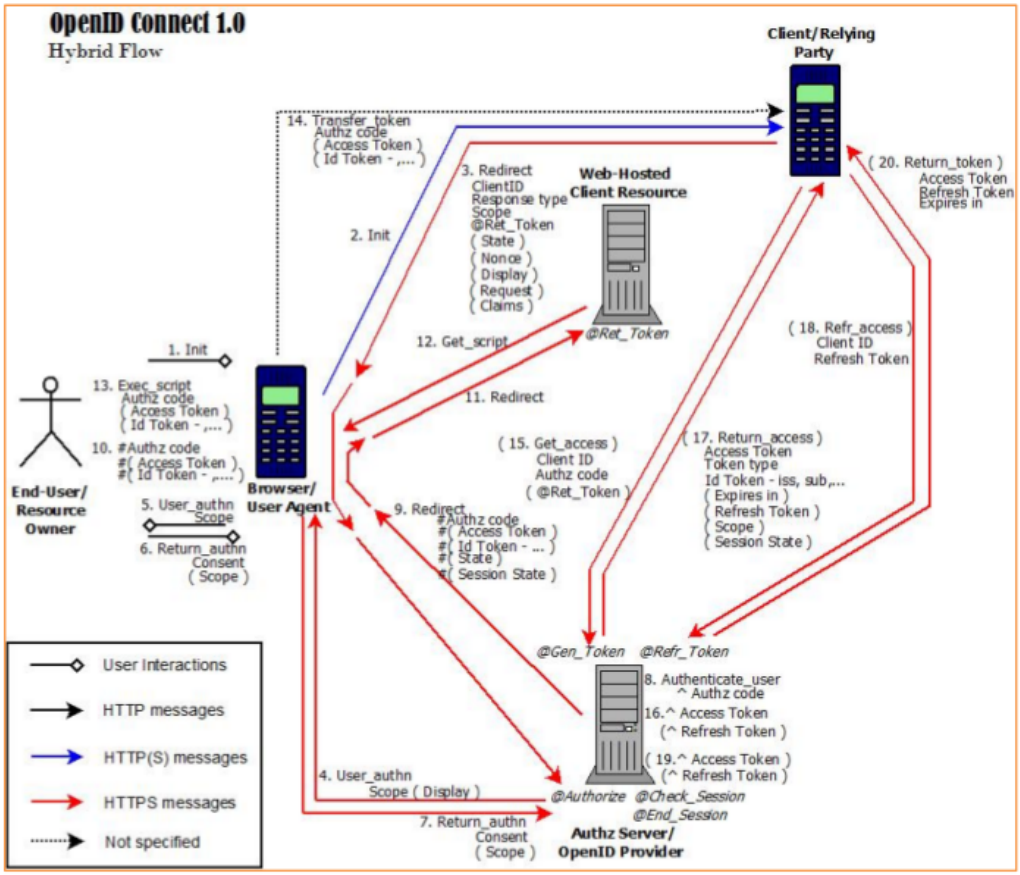

FIGURE 4.9 – OpenID Connect Hybrid flow (Source : [15])

4.2.4 JavaScript Object Signing and Encryption (JOSE)

L'outil JOSE fait parti des mécanismes de sécurité permettant d'atteindre les objectif de sécurité telles que la confidentialité et l'intégrité.

C'est un ensemble de projets standard de l'IETF pour permettre la protection cryptographique des objets JSON [19]. c'est un framework destiné à fournir une méthode pour transférer de manière sécurisée des réclamations "claims" (telles que des informations d'autorisation) entre les parties. L'outil JOSE fournit une collection de spécifications telle que le JSON Web Key, (JWK), le JSON Web Signature (JWS), le JSON Web Encryption (JWE),le JSON Web Algorithms JWA à cet effet.

JOSE exécute des fonctions similaires aux standards XML Signature et XML Encryption, bien qu'il existe des différences notables. Les normes cryptographiques XML dépendent de

l'utilisation des certificats X.509 délivrés par des Autorités de certification (CA) de confiance. JWK prend en charge l'utilisation des certificats X.509, mais ne l'exige pas. Les signatures XML exigent également que le document XML soit converti en une forme canonique standard avant de calculer la signature, mais JWS adopte l'approche plus simple de la signature d'un objet JSON dans la forme exacte dans laquelle il sera envoyé sur le réseau [19]

JWK

JSON Web Key est une structure de données représentant une clé cryptographique avec les données cryptographiques et d'autres attributs, tels que l'utilisation des clés.


```
{"kty":"EC",
 "crv":"P-256",
 "x":"f83OJ3D2xF1Bg8vub9tLe1gHMzV76e8Tus9uPHvRVEU",
 "y":"x_FEzRu9m36HLN_tue659LNpXW6pCyStikYjKIWI5a0",
 "kid":"Public key used in JWS spec Appendix A.3 example"
}
```


FIGURE 4.10 – Exemple de clé JWK (Source : [12])

JWS

JSON Web Signature (JWS) représente le contenu sécurisé avec les signatures numériques ou les codes d'authentification des messages (MAC) à l'aide des structures de données [RFC7159] basées sur JSON. Les mécanismes cryptographiques JWS offrent une protection intégrale pour une séquence arbitraire d'octets [13]. Il offre la signature de données sous formes d'objet JSON, permettant donc d'assurer l'intégrité des données.


```
{"iss":"joe",
 "exp":1300819380,
 "http://example.com/is_root":true}
```


FIGURE 4.11 – Exemple de JWS (Source : [13])

Le codage de cette charge utile JWS en tant que BASE64URL (JWS Payload) donne cette valeur (avec des sauts de ligne uniquement à des fins d'affichage).


```
eyJpc3MiOiJqb2UiLA0KICJleHAiOjEzMDA4MTkzODAsDQogImh0dHA6Ly9leGFt
cGxlLmNvbS9pc19yb290Ijp0cnVlfQ
```


FIGURE 4.12 – Données en BASE64URL d'un JWS (Source : [13])

JWE

JSON Web Encryption (JWE) représente le contenu crypté à l'aide de structures de données basées sur JSON [RFC7159]. Les mécanismes cryptographiques JWE chiffrent et fournisse une protection intégrale pour une séquence arbitraire d'octets [14]. Il permet donc de pouvoir chiffrer les données sous formes d'objet JSON permettant donc d'assurer la confidentialité des données.

Par défaut, pour chaque message, une nouvelle clé de chiffrement de contenu (CEK) devrait être générée. Cette clé est utilisée pour chiffrer le texte en clair et est rattachée au message final. La clé publique du destinataire ou une clé partagée est utilisée uniquement pour chiffrer le CEK (à moins que le cryptage direct ne soit utilisé, voir ci-dessous). Seuls les algorithmes AEAD (Authenticated Encryption with Associated Data) sont définis dans la norme, de sorte que les utilisateurs ne doivent pas penser à la façon de combiner JWE avec JWS

FIGURE 4.13 – Exemple de JWE (Source : [13])

L'encodage de cet en-tête protégé JWE comme BASE64URL (UTF8 (JWE Protected Header)) donne cette valeur.

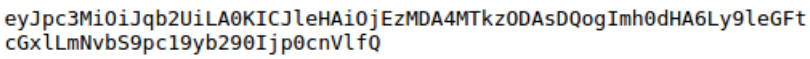

FIGURE 4.14 – Données en BASE64URL d'un JWS(Source : [13])

JWA

JSON Web Algorithms définit les algorithmes et leurs identifiants à utiliser dans le JWS et le JWE. Les trois paramètres qui spécifient les algorithmes sont "alg" pour JWS, "alg" et "enc" pour JWE.

Dans ce chapitre nous avons présenté d'une part les différents efforts fournis pour assurer la sécurité niveau message pour les API Restful et d'autre part nous avons mis en évidence les quelques outils libres les plus utilisés pour assurer la sécurité des API Restful.

Chapitre 5

Analyse Comparative Fonctionnelle

Ce chapitre est consacré à la comparaison des différentes solutions proposées aux deux chapitres précédents. D'une part nous comparons les solutions permettant d'assurer l'identité du client et du service et d'autre part des solutions liées à la confidentialité et à l'intégrité.

5.1 Comparaison des solutions d'authentification et/ou d'autorisation

Dans cette section nous avons décidé de comparer deux à deux les solutions permettant d'assurer l'autorisation et/ou l'authentification.
Un tableau comparatif fournissant un aperçu de notre analyse est présenté à la fin de cette section.

5.1.1 OAuth 2.0 vs OpenID Connect

Bien qu'ayant des différences, les protocoles OAuth 2.0 et OpenID Connect présentent certains aspects en commun parmi lesquels on peut citer l'utilisation des principes REST pour la communication et JSON comme format de message, le focus sur l'utilisateur (user-centriste) ou encore la possibilité de gérer les droits donnés au fournisseur de service (service provider).

Aussi OpenID Connect base son fonctionnement sur deux flows (implicit et autorisation code) qu'il reprend de OAuth 2.0. En terme de sécurité les deux protocoles exigent que les interactions soient effectuées sur un canal sécurisé en utilisant TLS à cause des informations échangées qui sont en texte claire [8][20].

Bien que cela soit une exigence du côté de OpenID Connect et une option du côté de OAuth 2.0 qui ne spécifie pas comment enregistrer le client[15], les deux protocoles offrent l'enregistre-

ment dynamique des clients , permettant ainsi aux nouveaux clients de s'inscrire en fournissant et en recevant des informations via le point d'inscription "registration endpoint".

La principale différence entre OAuth 2.0 et OpenID Connect réside dans le fait que OpenID Connect offre à la fois l'autorisation et l'authentification et que OAuth 2.0 quant à lui permet juste d'assurer l'autorisation [15], ce qui justifie l'utilisation du "ID token" introduit par OpenID Connect.

Il faut aussi noter que OAuth 2.0 permet le partage de données entre deux applications et OpenID Connect ne le permet pas [10]. En effet En addition aux deux flows repris de OAuth 2.0, OpenID Connect introduit l' "Hybride flow" qui est une combinaison du "Implicit flow" et du "Code Authorization flow" héritant ainsi certaines caractéristiques tels que les que retour de tokens au client comme dans l'implicit flow ou encore le fait que le client doit gérer la communication avec le service comme dans le "Code Authorization flow".

OpenID Connect à pour objectif principal d'offrir l'authentification unique "Signle Sign-On" pour applications clientes tandis que OAuth 2.0 offre l'autorisation d'API entre applications.

5.1.2 OpenID Connect vs SAML

OPenID Connect et SAML sont tous deux des protocoles qui offrent à la fois l'autorisation et l'authentification [27], tout en utilisant des différentes techniques.

Bien que OpenID Connect et SAML puissent être utiliser pour les mêmes objectifs c'est à dire assurer l'authentification et l'autorisation , les deux protocoles n'ont pas pour cible le même public. SAML se focalise sur les entreprises autrement dit il est **"organisation-centric"** tandis que OpenID Connect a pour focus les utilisateurs donc **"user-centric"**. Il faut noter aussi que SAML a été conçu uniquement pour les applications web, alors que OpenID Connect support aussi bien les applications web que les applications natives et les applications mobiles [10]

OpenID Connect est un protocole basé sur l'utilisation des techniques REST et JSON contrairement à SAML qui de son côté utilise du XML et du SOAP. Contrairement au XML dont le traitement (parsage) est lourd et coûteux en performance, le standard REST et le JSON sont simple à implémentés dans les langages les plus utilisés du web facilitant ainsi leur adoption. User consent ?

Pour l' échange des informations relatives à la sécurité SAML utilise une assertion qui est un document XML signé contenant des informations sur le sujet (qui a authentifié), les attributs (informations sur la personne), l'émetteur (qui a émis l'affirmation) et d'autres informations d'authentification [11]. L'équivalent dans OpenID Connect est l' "ID_TOKEN". Il s'agit d'un

document JSON signé précisément un JSON Web Token (JWT) qui contient le sujet, l'émetteur et les informations d'authentification [20]. Il faut aussi noter que ces éléments (token) permettant l'échange d'informations ont une durée de vie "expiration" aussi bien du côté de SAML que celui de OpenID Connect.

OpenID Connect exige l'utilisation de la sécurité niveau transport pour toutes interactions ainsi TLS/SSL doit être implémenter . La sécurité niveau message est optionnelle et laisser au soins des développeurs. cette sécurité niveau message peut être possible en utilisant du JWT. Quant à SAML la sécurité est implémenté niveau message en utilisant du XML Encryption et XML Signature, et TLS/SSL peut être implémenter en complément pour assurer la sécurité niveau transport.

OpenID Connect permet l'enregistrement dynamique de client "Dynamic Client Registration" par lequel les nouveaux clients peuvent s'inscrire en échangeant des informations avec l' "Identity Provider". SAML de son côté utilise une configuration qui se fait de façon statique. Aussi la découverte de l'Identity Provider se fait de façon dynamique du côté de OpenID Connect tandis qu'il est statique du côté de SAML.

Les deux protocoles ont comme objectif d'offrir l'authentification unique "SSO", SAML pour les entreprises et OpenID Connect pour les utilisateurs. SAML permet offre donc l'authentification unique pour les entreprises et OpenID connect pour les utilisateurs.

5.1.3 OAuth 2.0 vs SAML

OAuth 2.0 est un framework qui offre uniquement l'autorisation même si celle ci nécessite un processus d'authentification qui n'est pas du ressort de OAuth 2.0 [8]. SAML comme expliqué plus haut assure aussi bien l'authentification que l'autorisation.

En terme de tokens OAuth 2.0 utilise deux types de token pour assurer ses services, un "Access Token" qui permet d'accéder aux ressources protégées et un "Refresh Token" qui permet d'obtenir un nouveau "Access Token" en cas d'expiration ou invalidité de ce dernier [8]. Le protocole OAuth 2.0 ne spécifie pas un format spécifique pour le token, il peut être aussi bien du XML, que du JSON ou encore du binaire. SAML de son côté utilise plutôt des assertions en format XML contenant des informations sur le sujet, les attributs, l'émetteur et d'autres informations d'authentification.

Les deux protocoles ont aussi une différence au niveau du public cible. OAuth 2.0 se focalise sur les utilisateurs donc "user-centric" et SAML se concentre sur les entreprises donc "organization-centric". Ainsi SAML a pour but de permettre l'authentification unique (SSO) côté entreprise et OAuth 2.0 de permettre l'autorisation d'API entre applications.

Concernant le transport, les deux protocoles utilisent du HTTP, néanmoins SAML définit des liaisons (bindings) spécifiques tels que le HTTP Redirect et HTTP POST binding qui sont utilisées par le fournisseur d'identité pour rediriger l'utilisateur vers le fournisseur de services.

La sécurité au niveau de OAuth repose sur celle du canal de transport, donc l'utilisation du SSL/TLS. La sécurité du côté de SAML peut être assurer aussi bien en utilisant du SSL/TLS qu'en utilisant des outils tels que le chiffrement XML ou la signature XML.

OAuth à pour objectif de permettre l'autorisation d'API entre les applications tandis que SAML à pour but d'offrir l'authentification unique "SSO", pour les entreprises. Les configurations tels que l'enregistrement du client ou encore la découverte de l'IdP sont statique du côté de SAML alors que OAuth ne spécifie pas comment effectuer l'enregistrement du client, il peut être dynamique, via un formulaire HTTP etc... [15].

5.1.4 Tableau comparatif

Au terme de cette analyse comparant deux à deux les caractéristiques des différents protocoles permettant d'assurer l'identité des utilisateurs et des services nous résumons les observations dans le tableau ci-dessous pour en avoir une vue globale.

TABLE 5.1 – Tableau comparatif des solutions d'autorisation et d'authentification

Aspects	**SAML 2.0**	**OpenID Connect**	**OAuth 2.0**
Focus (centicity)	Entreprise	Utilisateur	Utilisateur
Format message	XML	JSON	JSON, XML, Binaire, Autres
Token	SAML assertion	Access Token, ID Token	Access Token, Refresh Token
Expiration Token	Oui	Oui	Optionnelle
Transport	HTTP, HTTP POST Binding, HTTP REDIRECT Binding	HTTP GET, HHTP POST	HTTP
Flows	/	Autorization Code, Implicit, Hybrid	Autorization Code, Implicit, RO password credentials, Client credentials
Sécurité	XML Encryption, XML Signature, TLS/SSL	TLS/SSL	TLS/SSL
Enregistrement du client	Statique	Dynamique	Non spécifié (formulaire HTTP, dynamique ...)
Découverte IdP	Statique	Dynamique	/
Fonctionnalité(s)	Authentification, Autorisation	Authentification, Autorisation	Autorisation
Utilisation dédiée	SSO pour les entreprises	SSO pour les applications clientes	Autorisation d'API entre applications
Consentement de l'utilisateur	Non	Oui	Oui

5.2 Comparaison des solutions de confidentialité et d'intégrité

Il s'agit dans cette section de comparer deux à deux les solutions de sécurité des API qui permettent d'assurer la confidentialité et l'intégrité qui permet d'assurer ainsi la non-répudiation . Un tableau comparatif offrant une vue globale sur l'analyse est présenté pour clôturer cette

section.

5.2.1 TLS/SSL vs WS-Security

Le protocole SSL/TLS et le WS-Security sont des outils qui offrent la confidentialité et l'intégrité. Le protocole SSL/TLS dans son fonctionnement nécessite que le serveur et optionnellement le client prouvent leur identité par l'utilisation des clés publiques dont chaque partie s'assure de la validité, ainsi SSL/TLS fournit aussi l'authentification.

La sécurité fournie par SSL/TLS s'effectue au niveau transport, donc en protégeant le canal de communication emprunter par les données lors des échanges entres les différentes parties. Ce type de sécurité appelé "point-to-point security" protèges les données seulement d'un nœud à un autre, dans le cas d'utilisation d'un proxy par exemple, les données ne sont plus protégées entre le proxy et le server. Le WS-Security de son côté, plutôt que de sécuriser le canal de communication, il applique la sécurité au niveau du message, fournissant ainsi la sécurité "end-to-end".

Le Standard WS-Security comme tous standard définit pour les Web Services SOAP, est basé sur du XML, les données sécurisées par ce dernier sont alors dans une synthaxe XML nécessitant ainsi des traitement qui peuvent coûter en performance. Du côté du protocole TLS/SSL les informations sécurisé sont dans un format "plain-text" .

Le WS-Security offre la possibilité de chiffrer une partie du contenu ou alors le contenu au complet [22]. Cette option n'est pas fournit par le protocole SSL/TLS, les opérations de chiffrement s'applique à tous le contenu.

Le WS-Security définit des tokens profiles qui décrivent comment certains type de tokens peuvent être utiliser comme informations de sécurité (certificat, assertions) dans les messages SOAP. Du côté de TLS/SSL les informations (certificat, secret partager) de sécurité sont définis lors de l'étape d'authentification.

Pour assurer la confidentialité et l'intégrité des données le WS-Security utilise le chiffrement et la signature XML qui peuvent utiliser aussi bien des algorithmes de cryptographie symétrique que algorithmes de cryptographie asymétrique. Le SSL/TLS ne nécessitant pas un format de données spécifique n'a pas recours aux outils tels que le XML Encryption ou le XML Signature,cependant il n'utilise que des algorithmes de cryptographie symétrique pour sécuriser les données car ils sont plus rapide que les algorithmes de cryptographie asymétrique.

5.2.2 TLS/SSL vs JSON Object Signature & Encryption

Comparer l'outil JOSE au protocole TLS/SSL en terme de fonctionnalités et de niveau de sécurité revient à comparer TLS/SSL et WS-Security sur ces mêmes aspects. Ainsi JOSE fournit la confidentialité et l'intégrité tandis que TLS/SSL fournit en plus l'authentification. L'outil JOSE fournit la sécurité au niveau message alors que TLS/SSL assure la sécurité au niveau transport.

En terme de format de données, l'outil JOSE utilise du JSON tandis que le protocole TLS/SSL utilise du "plain-text" c'est à dire que les données chiffrées sont dans leur forme brute et non dans une syntaxe spécifique. Le deux outils appliquent leur mécanismes de sécurité sur la totalité du contenu, ils ne permettent pas de le faire seulement sur une partie du contenu.

En terme de token pour définir les informations de sécurité, TLS/SSL utilise des certificat et des secret partagé de sécurité sont initiés lors de l'étape d'authentification. Du côté de JOSE ces informations de sécurité sont décrit à l'aide du JSON Web Key. L'outil JOSE utilise aussi bien des algorithmes de cryptographie symétrique que des algorithmes de cryptographie asymétrique tandis que le TLS/SSL n'utilise que des algorithmes de cryptographie symétrique pour sécuriser les données.

Pour assurer la sécurité au niveau messages ,l'outil JOSE définit le JWE et le JWS pour assurer respectivement la confidentialité et l'intégrité. Il définit aussi le JSON Web Algorithms (JWA) pour spécifier les algorithmes qui sont supportés par le JWE et le JWS pour assurer leur fonctionnalités, TLS/SSL quant à lui ne définit pas de tels mécanismes du au fait qu'il n'utilise pas une syntaxe spécifique.

5.2.3 WS-Security vs JSON Object Signature & Encryption

Le JSON Object Signature & Encryption et le WS-Security sont tout les deux des solutions qui fournissent la confidentialité et l'intégrité, mais en utilisant des moyens différents.

Les deux solutions assure la sécurité au même niveau, c'est à dire au niveau message, offrant ainsi la sécurité "End-to-End" en d'autre terme la protection des données peu importe le nombre de nœud par lesquels passent ces dernières pour arriver chez le destinataire.

En terme de format de données, le WS-Security utilise du XML tandis que JOSE utilise du JSON. Il faut aussi notifier que le WS-Security permet de sécuriser une partie ou tout le contenu du message alors que le JOSE n'offre que la protection du contenu en entier.

Le WS-Security des profiles qui définissent des moyens par lesquels des tokens peuvent être utiliser dans les messages SOAP afin d'assurer leur sécurité. Du côté de JOSE des informations de sécurité telles que les clés, les certificats ou encore des secrets partagés sont définit par l'utilisation des JSON Web Key (JWK).

Pour assurer la sécurité au niveau messages ,le WS-Security utilisent les technologies XML Encryption et XML Signature permettant d'offrir respectivement la confidentialité et l'intégrité. JOSE de son côté définit le JWE et le JWS pour les mêmes objectifs. Il définit aussi le JSON Web Algorithms (JWA) pour spécifier les algorithmes qui sont supportés par le JWE et le JWS pour assurer leur fonctionnalités.

5.2.4 Tableau comparatif

Pour conclure de cette analyse comparant deux à deux les caractéristiques des différentes solutions permettant d'assurer la confidentialité et l'intégrité offrant aussi la non-répudiation, nous résumons les observations dans le tableau ci-dessous pour en avoir une vue globale.

TABLE 5.2 – Tableau comparatif des solutions de confidentialité et d'intégrité

Aspects	**SSL/TLS**	**WS-Security**	**JOSE**
Fonctionnalité(s)	Authentification, Confidentialité Intégrité	Confidentialité Intégrité	Confidentialité Intégrité
Niveau de sécurité	Transport	Message	Message
Type de sécurité	Point-to-Point	End-to-End	End-to-End
Format message	Texte	XML	JSON
Contenu sécurisé	Complet	Partiel/Complet	Complet
Token Profile /Token	Certificat, Secret partagé	Username token profile, X.509 certificate token profile, Kerberos token profile, SAML token profile, XrML/REL token profile	JSON Web Key
Type d'algorithme cryptographique appliquée aux données	Symétrique	Symétrique, Asymétrique	Symétrique, Asymétrique
Outils	/	XML Encryption, XML Signature	JSON Web Encryption, JSON Web Signature, JSON Web Algorithms,

l'identité des utilisateurs et des services

Ce chapitre a fait l'objet d'une comparaison des outils permettant de sécuriser les api web, d'une part nous avons compareé deux à deux les outils offrant l'authentification et l'autorisation et d'autre part les solutions permettant d'assurer la confidentialité, l'intégrité et la non-répudiation.

Chapitre 6

Exemple d'application

Dans ce chapitre nous présentons (développons) un système de référence démontrant l'utilisation de ces outils permettant de sécuriser des API Web, mettant ainsi en évidence l'intérêt et l'importance de les intégrer dans nos API. L'adoption des Restful Services étant en croissance de nos jours nous avons donc opté pour ce type de service comme exemple d'application.

6.1 Scénario

Notre but n'étant pas d'implémenté les spécifications définies par les méthodes de sécurité que nous avons précédemment étudié, mais plutôt de montrer leur utilisation à travers un outils de référence. Ainsi pour notre exemple nous avons choisi d'utiliser l'implémentation de OAuth 2.0 défini par Google. Il faut noter que cette implémentation de OAuth 2.0 de Google est conforme à la spécification de OpenID Connect, donc en utilisant celle-ci nous montrons par la même occasion l'utilisation de la spécification OpenID Connect. Aussi Google Oauth 2.0 Signe le "ID Token" renvoyé aux clients avec un JWK, pour assurer l'intégrité de ce dernier. Ainsi les applications clientes peuvent vérifier l'intégrité de celle-ci,de cette façon nous démontrons aussi l'utilisation de la spécification JOSE qui est utilisé pour la signature et la vérification.

Grâce à ce serveur Google permet à des applications clientes d'accéder à ses divers API telles Google Calendar API, Google Drive API et des dizaines d'autres. Notre objectif étant une simple démonstration du mécanisme de fonctionnement de certains des outils de sécurité étudiés, comme exemple d'application nous avons décidé de développer un application web java dont l'accès est autorisé qu'en s'authentifiant avec un compte google. Ainsi dans notre scénario Google joue le rôle de serveur d'autorization et de ressource, notre application joue le rôle de client et l'utilisateur voulant accéder à notre application celui du "resource owner" propriétaire de la ressource.

Nous avons décidé pour notre système d'utilisé la librairie JAX-RS qui permet de développer des services Restful en Java. Pour le client OAuth 2.0 nous avons utilisé l'implémentation d'apache oltu,pour l'outil JOSE nous utilisons la librairie jose4j.

6.2 Résultats de l'implémentation

Comme expliquer dans les chapitres précédents, pour utiliser le serveur d'autorisation, le client doit s'enregistrer auprès de ce dernier. Google fourni une plate-forme pour enregistrer les clients . Cela à consisté à la première étape de notre implémentation.

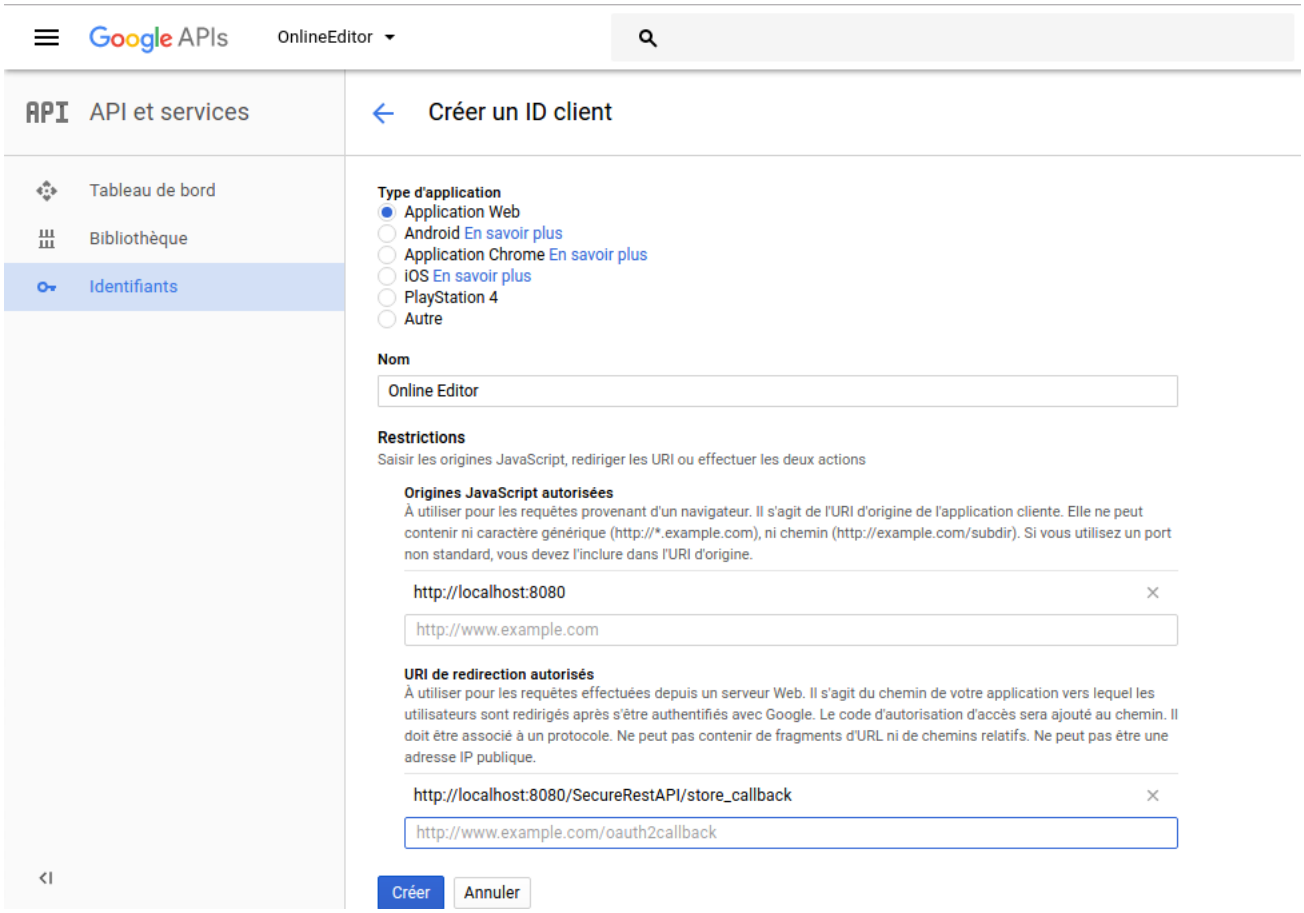

FIGURE 6.1 – Enregistrement du client

La figure ci-dessus présente un formulaire nous permettant de spécifier à google les uri de redirection à utiliser lors des requêtes d'autorisations. Une fois ces informations fournies et validées google nous génère un identifiant client et un secret comme le montre la figure ci-après. Ces informations seront utilisées lors des échanges entre notre application cliente et le serveur OAuth 2.0 de google.

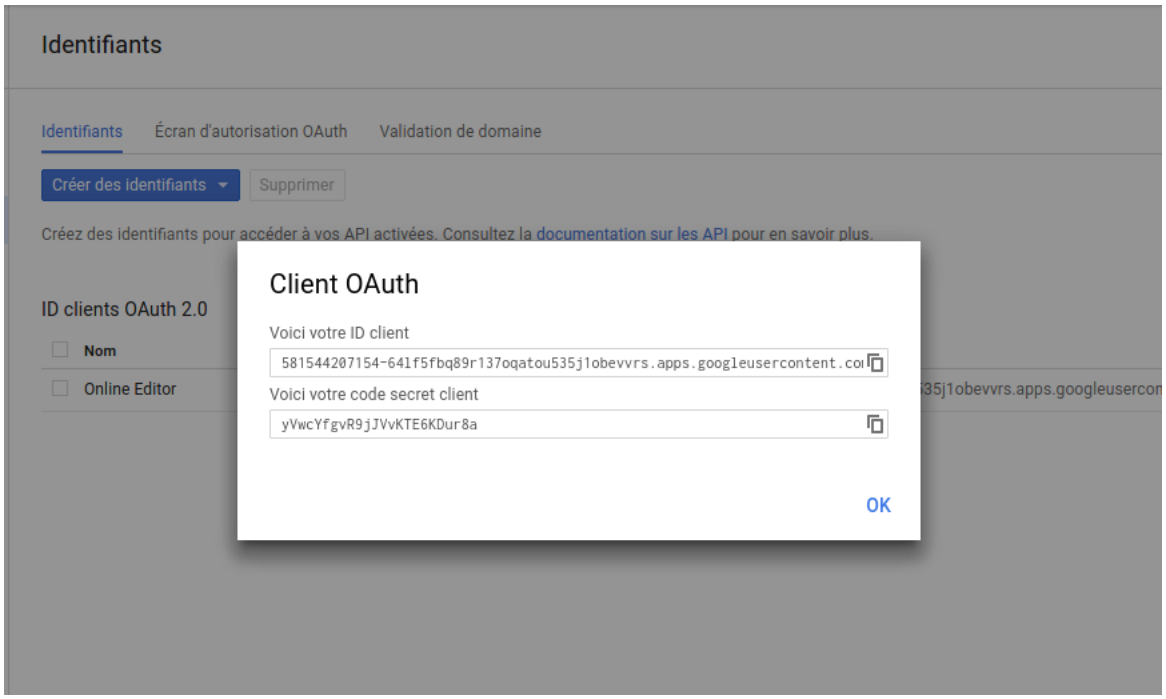

FIGURE 6.2 – Crédentials du client

A présent que notre application cliente est enregistrée auprès du serveur Oauth 2.0 de google, nous pouvons accéder à la fenêtre de login et procéder à l'authentification de l'utilisateur avec son compte google comme le montre la figure ci-après.

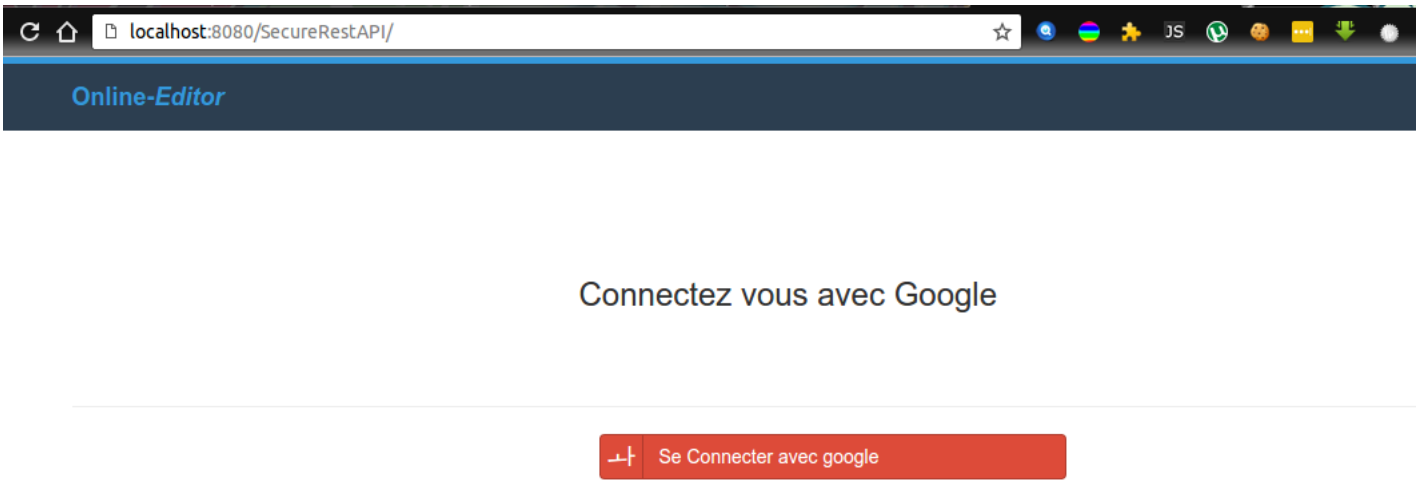

FIGURE 6.3 – Fenêtre d'Authentification du client

Lorsque l'utilisateur clique sur le bouton de connexion, il est redirigé vers google pour s'authentifier. Cette redirection est faite grâce à un Restful service de notre application cliente qui utilise les informations reçues lors de l'enregistrement pour contacter le serveur OAuth 2.0 de google.

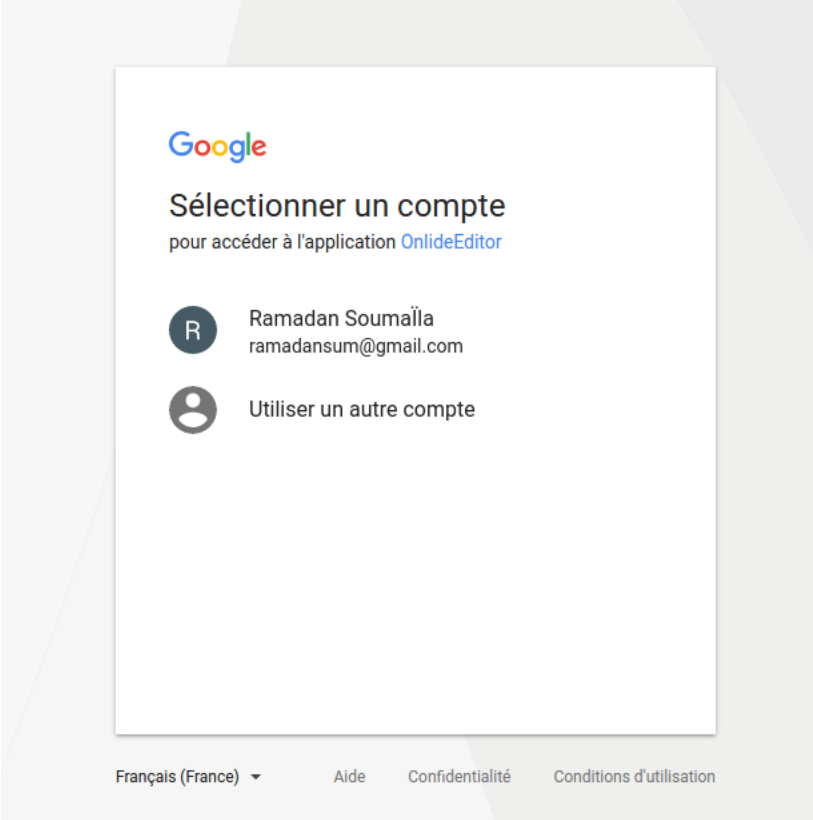

FIGURE 6.4 – Authentification de l'utilisateur sur Google

Une fois l'utilisateur authentifié avec Google, viens ensuite la phase d'autorisation pendant laquelle il autorise ou non notre application à accéder à sa ressource qui est dans ce cas son compte, précisément ses informations comme on peut le constater sur la figure suivante.

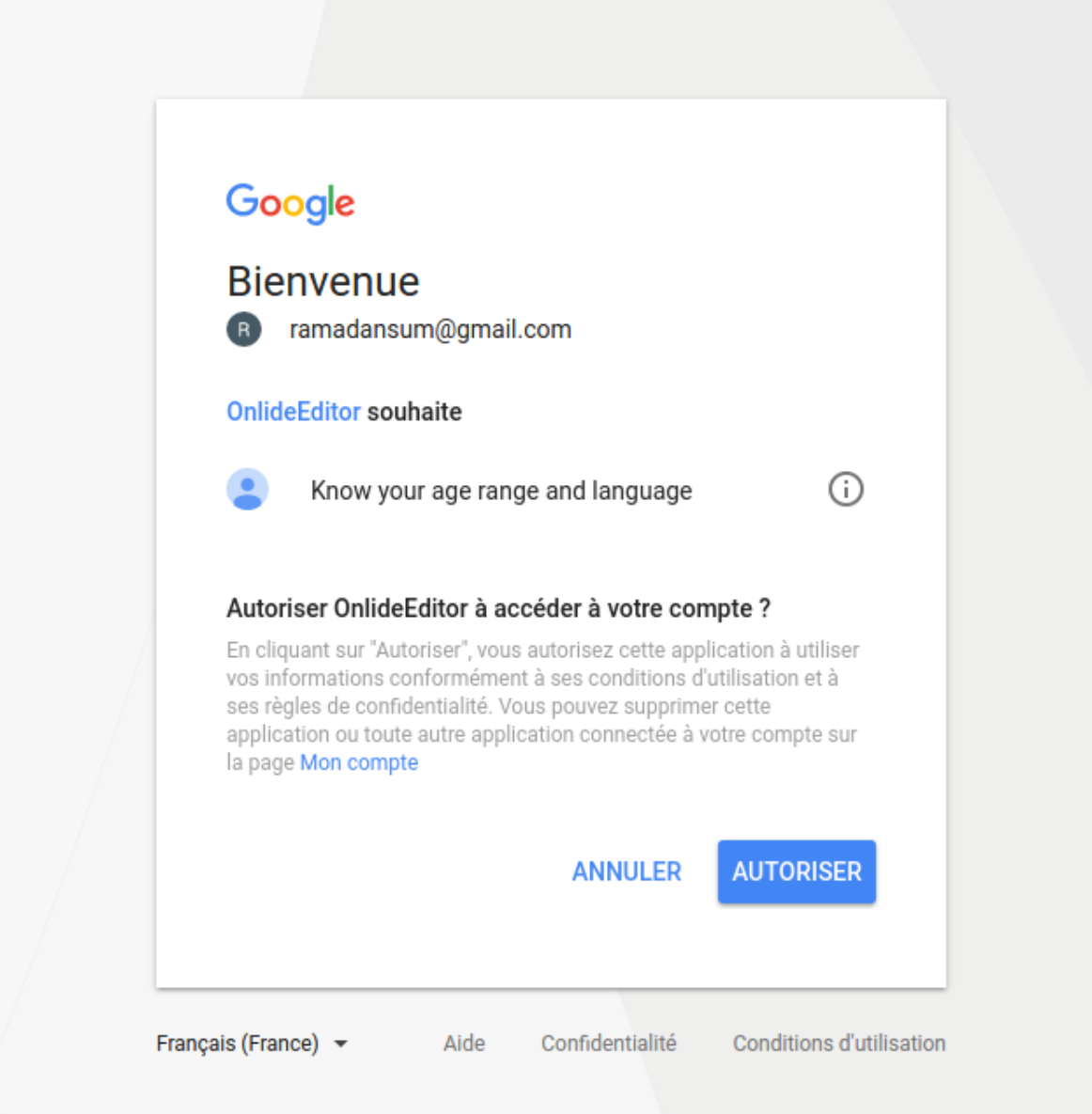

FIGURE 6.5 – Autorisation de l'utilisateur

Lorsque l'utilisateur autorise notre application à accéder à son compte, il est alors redirigé vers notre application cliente pour avoir l'accès. Lors de cette redirection google renvoi un code d'autorisation. Avant de donner l'accès à l'utilisateur, le code d'autorisation reçu par l'application cliente est utilisé pour demander un "token d'accès". Le serveur OAuth 2.0 de google renvoi alors au client des informations contenant non seulement le token d'accès mais aussi un "ID Token" qui démontre ainsi l'utilisation de la spécification OpenID Connect par le serveur OAuth 2.0 de google. Afin de s'assurer que le "ID Token" reçu est intègre, l'application cliente vérifie cela en utilisant la librairie jose4j montrant ainsi l'utilisation de la spécification JOSE.

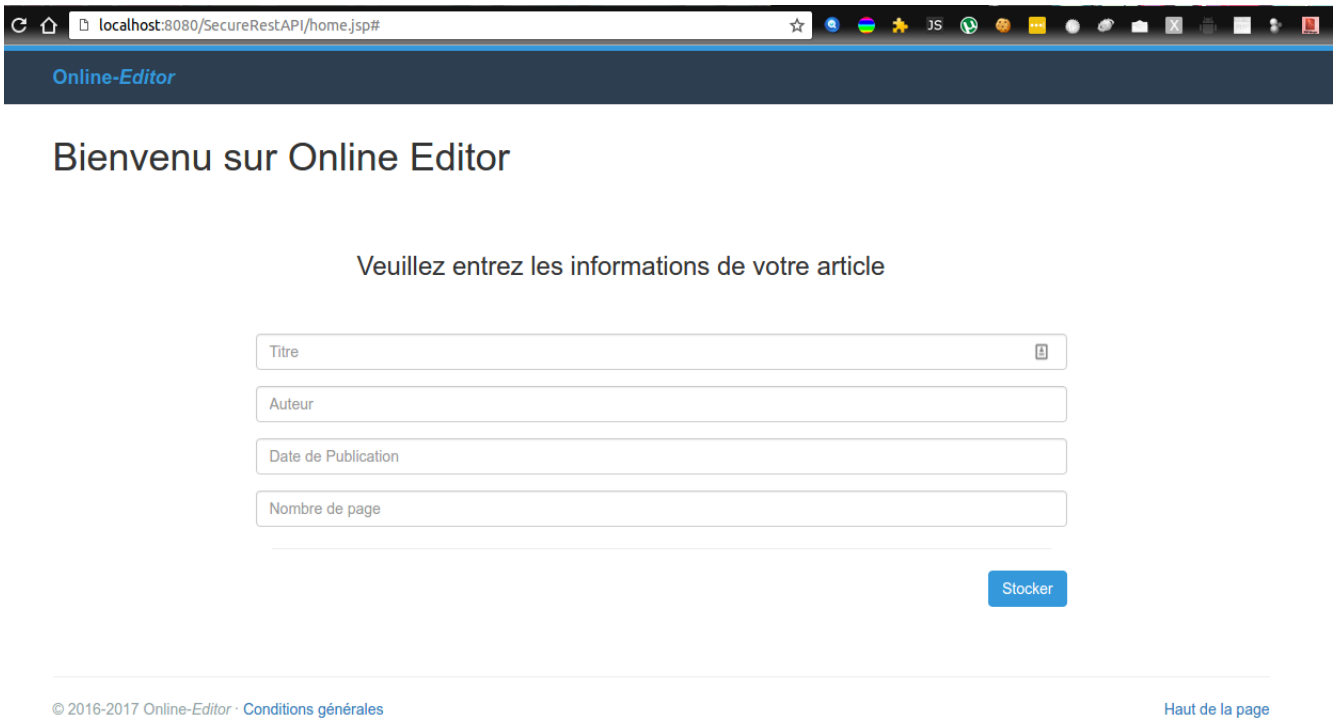

FIGURE 6.6 – Accueil application cliente

Dans ce chapitre nous avons présenté un exemple montrant le mécanisme de fonctionnement de certains outils implémentant des spécifications des méthodes de sécurité des API Web étudiées, d'une part nous avons abordé le scénario de l'exemple et d'autre part présenté les résultats de l'implémentation.

Conclusion

Les API Web, aussi bien les Web Services, que les Restful Services constituent le point central des infrastructures telles que des systèmes IoT (objets connectés), les plate-formes en ligne des entreprises commerciales et bien d'autres. Celles-ci échangent des données qui peuvent être sensibles, il est alors nécessaire que ces dernières soient sécurisées. De nombreux outils permettant de répondre à ce besoin existent, néanmoins trouver ou savoir quels sont les plus adéquats et plus avancés n'est pas toujours évident.L'objectif visé dans ce document était donc de faire une étude des techniques de sécurité des API Web les plus avancées, de comparer ces techniques et de montrer leur utilisation dans un système.

Nous avons donc commencé par présenter les deux types d'API Web en insistant sur leurs caractéristiques et principes que nous avons par la suite comparé brièvement. Nous avons montré par la suite les différents niveaux de sécurité qui peuvent être appliqués pour sécuriser ces API. Dans les chapitres 3 et 4 nous avons présenté respectivement les méthodes de sécurité des Web Services et celles des Restful Services en insistant sur leurs caractéristiques et leur fonctionnement.

Ensuite dans le chapitre 5 nous avons effectué une analyse comparative fonctionnelle des différentes techniques de sécurité les plus avancées et les plus utilisées pour les API Web. D'une part nous avons comparé deux à deux les techniques de sécurité permettant d'assurer l'authentification et l'autorisation et d'autre part celle offrant la confidentialité et l'intégrité et donc aussi la non-répudiation.Nous avons ainsi montré les similitudes et différences qui existent entre ces outils et les cas d'utilisations auxquels ils sont adaptés.

Enfin, nous avons montré dans le dernier chapitre l'utilisation de certaines de ces techniques de sécurité à travers un système que nous avons développez. Notre système constitué du serveur OAuth 2.0 de google jouant le rôle de serveur d'autorisation et de ressources et de d'une application cliente consommatrice de ressources, précisément des informations des utilisateurs. Pour assurer la sécurité de cette API, nous avons utilisé des librairies offrant des implémentations de OAuth 2.0, de OpenID Connect et de JOSE, celles-ci nous ont permis d'assurer l'iden-

tité du client et du service, d'assurer l'intégrité, l'authenticité et la non-répudiation des données échangées.

Notre analyse comparative des outils de sécurité s'est focalisée sur les aspects fonctionnels de ces derniers. En se limitant sur ces aspects nous n'avons pas pu explorer tous les angles sur lesquels ces outils peuvent être comparés. En effet en se basant sur ces aspects fonctionnels il n'est pas possible de comparer les performances de ces outils. Nous aurions pu implémenter tous ces outils de sécurité afin de comparer leur performance en cours d'exécution. Cela représente une piste intéressante qui peut faire l'objet de recherches ultérieures.

Bibliographie

[1] Gustavo Alonso, Fabio Casati, Harumi Kuno, and Vijay Machiraju. Web services. In *Web Services*, pages 123–149. Springer, 2004.

[2] Recep Bacı. Development of a web services security archhitecture based on. net framework. Master's thesis, İzmir Institute of Technology, 2008.

[3] Robert-Jan Boezeman. Web services security. Master's thesis, University of Nijmegen, Security of Systems (SoS) group and University of Oxford, 2003.

[4] Scott Cantor, Internet2 John Kemp, Nokia Rob Philpott, and E Maler. Assertions and protocols for the oasis security assertion markup language. *OASIS Standard (March 2005)*, 2005.

[5] Francisco Curbera, Matthew Duftler, Rania Khalaf, William Nagy, Nirmal Mukhi, and Sanjiva Weerawarana. Unraveling the web services web : an introduction to soap, wsdl, and uddi. *IEEE Internet computing*, 6(2) :86–93, 2002.

[6] Tim Dierks. The transport layer security (tls) protocol version 1.2. 2008.

[7] Roy T Fielding and Richard N Taylor. Principled design of the modern web architecture. *ACM Transactions on Internet Technology (TOIT)*, 2(2) :115–150, 2002.

[8] Dick Hardt. The oauth 2.0 authorization framework. 2012.

[9] Keiko Hashizume. *Web services cryptographic patterns*. Florida Atlantic University, 2009.

[10] Nick Heijmink, E Poll, and Rogier Hofboer. *Secure Single Sign-On*. PhD thesis, Master Thesis, 2015.

[11] John Hughes and Eve Maler. Security assertion markup language (saml) v2. 0 technical overview. *OASIS SSTC Working Draft sstc-saml-tech-overview-2.0-draft-08*, pages 29–38, 2005.

[12] Michael Jones. Json web key (jwk). 2015.

[13] Michael Jones, John Bradley, and Nat Sakimura. Json web signature (jws). Technical report, 2015.

[14] Michael Jones and Joe Hildebrand. Json web encryption (jwe). Technical report, 2015.

[15] O Manso, M Christiansen, and G Mikkelsen. Comparative analysis-web-based identity management systems. Technical report, Technical report, The Alexandra Institute, 2014.

[16] Standard OASIS. Web services security : Soap message security 1.1. *http ://www. oasis-open. org/committees/tc_home. php ? wg_abbrev= wss*, 2006.

[17] Pavan Kumar Potti. On the design of web services : Soap vs. rest. 2011.

[18] Alex Rodriguez. Restful web services : The basics. *IBM developerWorks*, 2008.

[19] M Russell. Secure restful interfaces : Business-oriented use cases & associated distributed security requirements.

[20] Nat Sakimura, J Bradley, M Jones, B de Medeiros, and C Mortimore. Openid connect core 1.0 incorporating errata set 1. *The OpenID Foundation, specification*, 2014.

[21] Gabriel Serme, Anderson Santana de Oliveira, Julien Massiera, and Yves Roudier. Enabling message security for restful services. In *Web Services (ICWS), 2012 IEEE 19th International Conference on*, pages 114–121. IEEE, 2012.

[22] Omar Slomic. A message-level security approach for restful services. Master's thesis, 2011.

[23] OASIS Standard. Ws-secureconversation 1.3, 2007.

[24] OASIS Standard. Ws-securitypolicy 1.2, 2007.

[25] OASIS Standard. Ws-trust 1.4. *DOI= http ://docs. oasis-open. org/ws-sx/ws-trust/v1*, 4, 2009.

[26] OASIS Standard. Web services federation language (ws-federation) version 1.2, 22 may 2009, 2010.

[27] Bas Zoetekouw and Martijn Oostdijk. Openid connect for surfconext, 2012.

Annexe A

Les 2 autres flows de OAuth 2.0

A.1 Resource Owner Password Credentials flow

Les informations d'authentification du propriétaire de ressource (c'est-à-dire, le nom d'utilisateur et le mot de passe) peuvent être utilisées directement comme un octroi d'autorisation pour obtenir un jeton d'accès. Ce flow devrait seulement être utilisé quand il y a un haut degré de confiance entre le propriétaire de ressource et le client (par exemple, le client fait partie du dispositif du système d'exploitation ou une application fortement privilégiée) et quand d'autres types de d'octroi d'autorisation ne sont pas disponibles (comme un code d'autorisation).

Bien que ce type d'octroi exige l'accès direct du client aux informations d'authentification du propriétaire de ressource, les références(la pièce d'identité) de propriétaire de ressource sont utilisées pour une requête unique et sont échangées contre un jeton d'accès. Ce type de d'octroi peut éliminer le besoin du client de stocker les informations d'authentification du propriétaire pour d'utilisation future, en fournissant un token d'accès ayant une longue durée de vie.

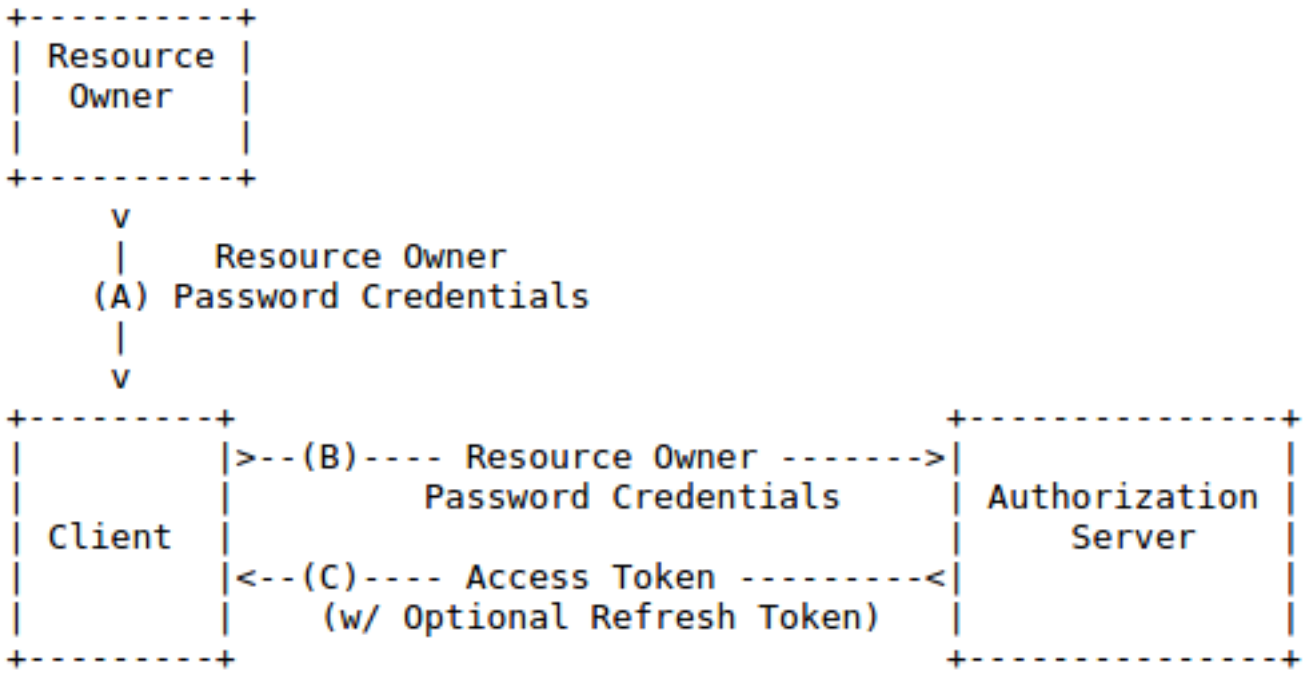

FIGURE A.1 – OAuth Resource Owner Password Credentials flow (Source : [8])

A.2 Client Credentials flow

Les informations d'authentification du client (ou d'autres formes d'authentification de client) peuvent être utilisées comme un octroi d'autorisation quand la portée d'autorisation est limitée aux ressources protégées sous le contrôle du client, ou aux ressources protégées dont l'autorisation a précédemment été arrangée avec le serveur d'autorisation. Les informations d'authentification du client sont utilisées comme un octroi d'autorisation typiquement quand le client agit en son propre compte (le client est aussi le propriétaire de ressource) ou demande d'avoir accès aux ressources protégées basées sur une autorisation précédemment arrangée avec le serveur d'autorisation.

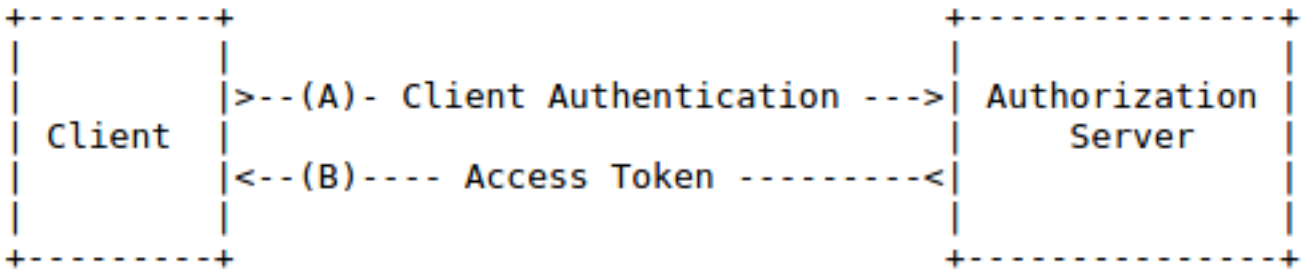

FIGURE A.2 – OAuth Client Credentials flow (Source : [8])