XML & Java

XML avec Java

- Objectifs
  - Traitement des fichiers XML en Java
  - Utilisation avec DOM
  - Utilisation avec SAX
  - Utilisation de XPath

Contenu

- Introduction
- DOM
- SAX
- XPath

Introduction

- XML (eXtensible Markup Language)
  - Standard W3C pour représenter des données dans des documents balisés
  - Définition **structurée** de l'information (syntaxe + sémantique)
  - Représentation **arborescente**
  - Possibilité de définir une **grammaire** (DTD)

Introduction

- JAXP (Java API for XML Processing)

  Capacité à lire et valider les documents XML, formé de 3 composants :

  – DOM (Document Object Model) → Arbre
  – SAX (Simple API for XML) → Evénement
  – StAX (Streaming API for XML) → entre DOM et SAX

Introduction

- StAX (Streaming API for XML)
  - –Peu gourmand en mémoire
  - –Accès séquentiel aux données mais contrairement à SAX on indique quel est le prochain élément à traiter
  - –Ecriture de fichiers XML
  - –StAX ne sera pas traité dans ce cours

DOM

- DOM (Document Object Model)
  - –API permettant d'accéder au contenu d'un document XML sous la forme d'une structure d'arbre
  - –Chargement complet du document XML
  - –Réservé à de petits fichiers
  - –Lecture linéaire

DOM

- DOM et Java
  - –Se référer aux package org.w3c.dom
  - –Le parser est issu de javax.xml.parsers.DocumentBuilder
  - – Obtenu depuis javax.xml.parsers.DocumentBuilderFactory

DOM

- Exemple de document XML

Exemple de travail

```
<?xml version="1.0" encoding="UTF-8"?>
<persons>
    <person id="1">
    <first-name>Donald</first-name>
    <last-name>Duck</last-name>
    <date-of-birth>01-01-1970</date-of-birth>
    <salary>1000.0</salary>
    </person>
    <person id="2">
    <first-name>Picsou</first-name>
    <last-name>Duck</last-name>
    <date-of-birth>02-01-1960</date-of-birth>
    <salary>120000.0</salary>
    </person>
    <person id="3">
    <first-name>Mickey</first-name>
    <last-name>Mouse</last-name>
    <date-of-birth>01-01-1980</date-of-birth>
    <salary>3000.20</salary>
    </person>
</persons>
```

DOM

- Création du parser

Exemple

```
1  // ouverture d'un document XML sous DOM
2  File fXmlFile = new File(fileName);
3  DocumentBuilderFactory dbFactory = DocumentBuilderFactory.newInstance();
4  // use DTD validation
5  dbFactory.setValidating(true);
6
7  DocumentBuilder dBuilder = dbFactory.newDocumentBuilder();
8  // report Errors if DTD validation is On
9  // note: ErrorHandler must be defined
10 dBuilder.setErrorHandler(new ErrorHandler());
11
12 Document doc = dBuilder.parse(fXmlFile);
13 doc.getDocumentElement().normalize();
```

DOM

- Lecture <tag>valeur</tag>

lecture <tag>value</tag>

```
1  NodeList nList;
2  // Obtain all nodes for last name
3  nList = doc.getElementsByTagName("last-name");
4
5  // treat each node
6  for (int i = 0; i < nList.getLength(); i++) {
7    Node nNode = nList.item(i);
8    if (nNode.getNodeType() == Node.ELEMENT_NODE) {
9      nNode.getTextContent().trim());
10   }
11 }
12
13
```

=> System.out.println(nNode.getTextContent().trim())

DOM

- Lecture <tag attr="value" />

lecture <tag attr="value" />

```
1  NodeList nList;
2  // get all nodes person
3  nList = doc.getElementsByTagName("person");
4  // treat each node
5  for (int i = 0; i < nList.getLength(); i++) {
6     Node nNode = nList.item(i);
7     if (nNode.getNodeType() == Node.ELEMENT_NODE) {
8        Element eElement = (Element) nNode;
9        // retrieve attribute id
10       String id=eElement.getAttribute("id").trim();
11       // note : attribute nationality is not defined in persons.xml
12       // so getAttribute will return an empty String
13       String nationality=eElement.getAttribute("nationality").trim();
14       if (nationality.isEmpty()) {
15          nationality="french";
16       }
17    }
18 }
```

Attribut inexistant : chaîne vide !

SAX

- SAX (Simple API for XML)
  - API permettant d'accéder au contenu d'un document XML de manière événementielle
  - Chargement partiel du document
  - Réservé à de gros fichiers
  - Lecture non linéaire

SAX

- SAX Handler
  - Définir une classe SAXReader qui hérite de org.xml.sax.helpers.DefaultHandler
  - Implémenter les méthodes de l'interface ContentHandler

SAX

- Méthodes de DefaultHandler

  Ce sont celles de ContentHandler, pour les principales :

  – startDocument

  – startElement

  – endElement

  – characters

  – endDocument

SAX

- Méthode characters

  Elle permet de lire une chaîne de caractères à l'intérieur d'une balise :

  <tag>value</tag>

  Ici on récupèrera : value

SAX

- Méthode startElement

  Elle permet de traiter les attributs :

  <tag attr="vattr">. . .</tag>

  En vérifiant qu'elle est la valeur de la balise tag

SAX

- Méthode endElement

  Elle permet de traiter :

  <tag>value</tag>

  En vérifiant qu'elle est la valeur de la balise tag

SAX

- Lecture <tag attr="vattr">value</tag>

Il faut combiner les deux méthodes :

- startElement pour lire les attributs et créer une structure de données
- Puis endElement pour lire value et l'affecter à la structure de données

SAX

- Création parser

Exemple

```
1  import javax.xml.parsers.*;
2  import org.xml.sax.*;
3  import org.xml.sax.helpers.*;
4
5  public class SAXReader  extends DefaultHandler {
6
7    protected String text;
8
9    public SAXReader()  throws Exception {
10     super();
11   }
12
13   public void read(String fileName) {
14     try {
15       XMLReader parser = XMLReaderFactory.createXMLReader();
16       parser.setContentHandler(this);
17       parser.setFeature("http://xml.org/sax/features/validation", true);
18       parser.parse(fileName);
19     } catch (Exception e) {
20       System.err.println(e.getMessage());
21     }
22   }
23
24   public void characters(char[] ch, int start, int length)
25       throws SAXException {
26     text = new String(ch, start, length);
27   }
28
```

SAX

- Lecture <tag>value</tag>

lecture <tag>value</tag>

```
1  // read <tag>value</tag>
2  public void endElement(String uri, String localName, String qName)
3     throws SAXException {
4
5     if (localName.equals("tag")) {
6
7        String value = text.trim();
8
9     }
10 }
11
```

SAX

- Lecture <tag attr="value" />

lecture <tag attr="value" />

```
1  // read <tag attr="vattr" />
2  public void startElement(String uri, String localName, String qName,
3     Attributes atts) throws SAXException {
4
5     if (localName.equals("tag")) {
6
7        String vattr = atts.getValue("attr").trim();
8
9        // note : be careful because atts.getValue will return null
10       // if attribute is not present
11       if (vattr==null) {
12          vattr="";
13       }
14    }
15
16 }
```

attribut inexistant : null !

XPath

- XPath (XML Path Language)
  - Il est à XML ce que SQL est aux SGBD relationnels
  - Langage de sélection de balises / attributs
  - Packages Java
    javax.xml.xpath.Xpath
    javax.xml.xpath.XPathFactory
    ou
    javax.xml.xpath.*

XPath

- Sans Xpath
  - Avec DOM, utiliser :
    - getElementByTagName
    - getElementByid
  - Avec SAX
    - startElement
    - endElement

XPath

- XPath et DOM

Avec DOM

```
1  // doc is a DOM Document
2  XPathFactory pathFactory=XPathFactory.newInstance();
3  XPath xpath = pathFactory.newXPath();
4  XPathExpression expr = xpath.compile("//person");
5
6  // DOM specific
7  NodeList nList = (NodeList)expr.evaluate(doc,XPathConstants.NODESET);
8
9  for (int i = 0; i < nList.getLength(); i++) {
10   Node nNode = nList.item(i);
11   if (nNode.getNodeType() == Node.ELEMENT_NODE) {
12     // your code here
13   }
14 }
```

Méthodes : getFirstChild, getNodeValue

XPath

- XPath et SAX

Avec SAX

```
1  // fileName is a String
2  File file=new File(fileName);
3  InputSource source = new InputSource(new FileInputStream(file));
4  XPathFactory fabrique = XPathFactory.newInstance();
5  XPath xpath = fabrique.newXPath();
6  XPathExpression expr = xpath.compile("//person");
7
8  // SAX specific
9  NodeList nList = (NodeList)expr.evaluate(source, XPathConstants.NODESET);
10
11 for (int i = 0; i < nList.getLength(); i++) {
12    Node nNode = nList.item(i);
13    if (nNode.getNodeType() == Node.ELEMENT_NODE) {
14       // your code here
15    }
16 }
```

XPath

- XPathExpression.evaluate(source,returnType)

Evaluer l'expression XPath en fonction du contexte

  - source : InputSource
  - returnType : QName, un parmi :
    - NODE, NODESET
    - NUMBER, BOOLEAN, STRING

Sécurité pour XML

Abdessamad Imine

Université de Lorraine & INRIA-LORIA Grand-Est

Abdessamad.Imine@loria.fr

Sommaire

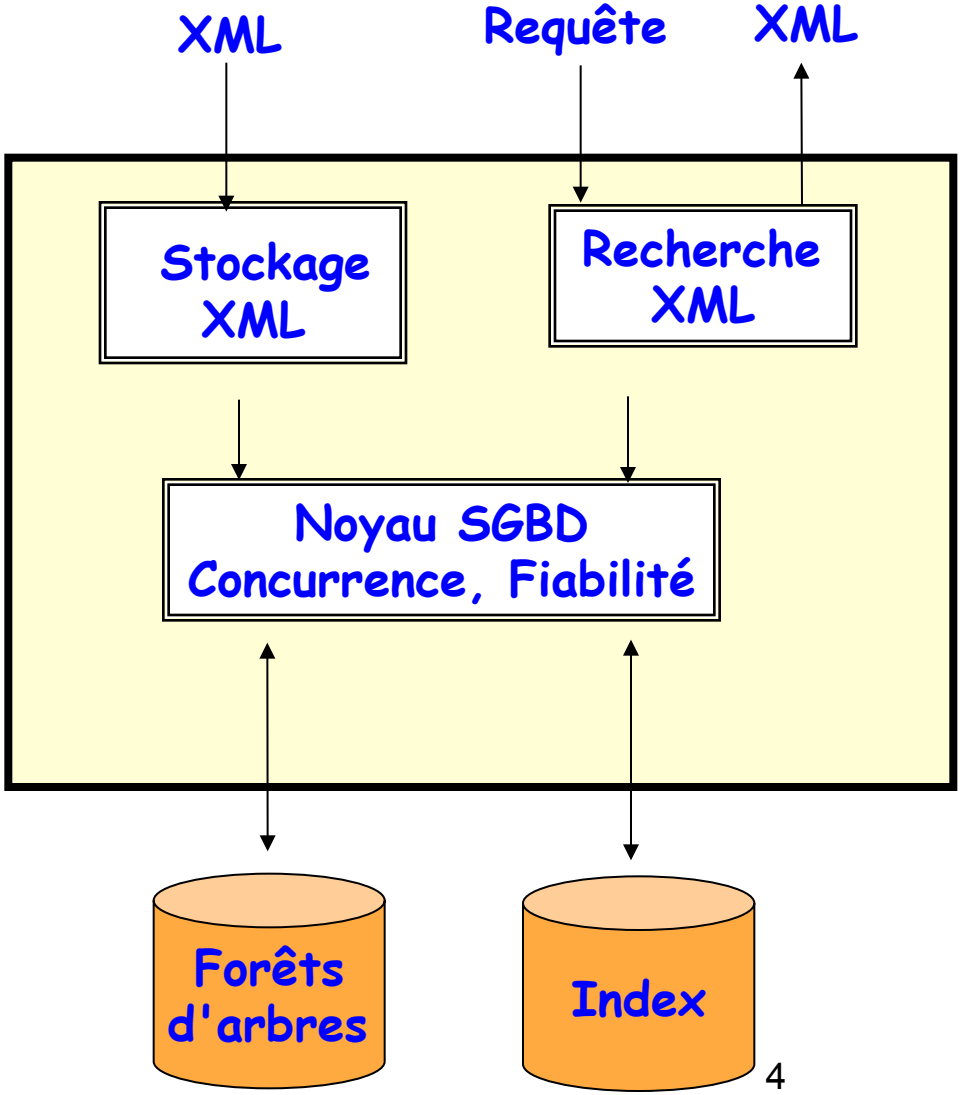
- Introduction
- Signature de XML
- Attaques contre XML

XML et BD

- Intégration des données et méta-données
- Standard d'échange de données universel
- Les BD ne peuvent rester indifférentes
  - nécessité de stocker les documents XML
  - nécessité de pouvoir interroger ces documents
  - évolution ou révolution ?
- Des efforts se font pour définir
  - un modèle de données
  - langage d'interrogation

XML et BD (2)

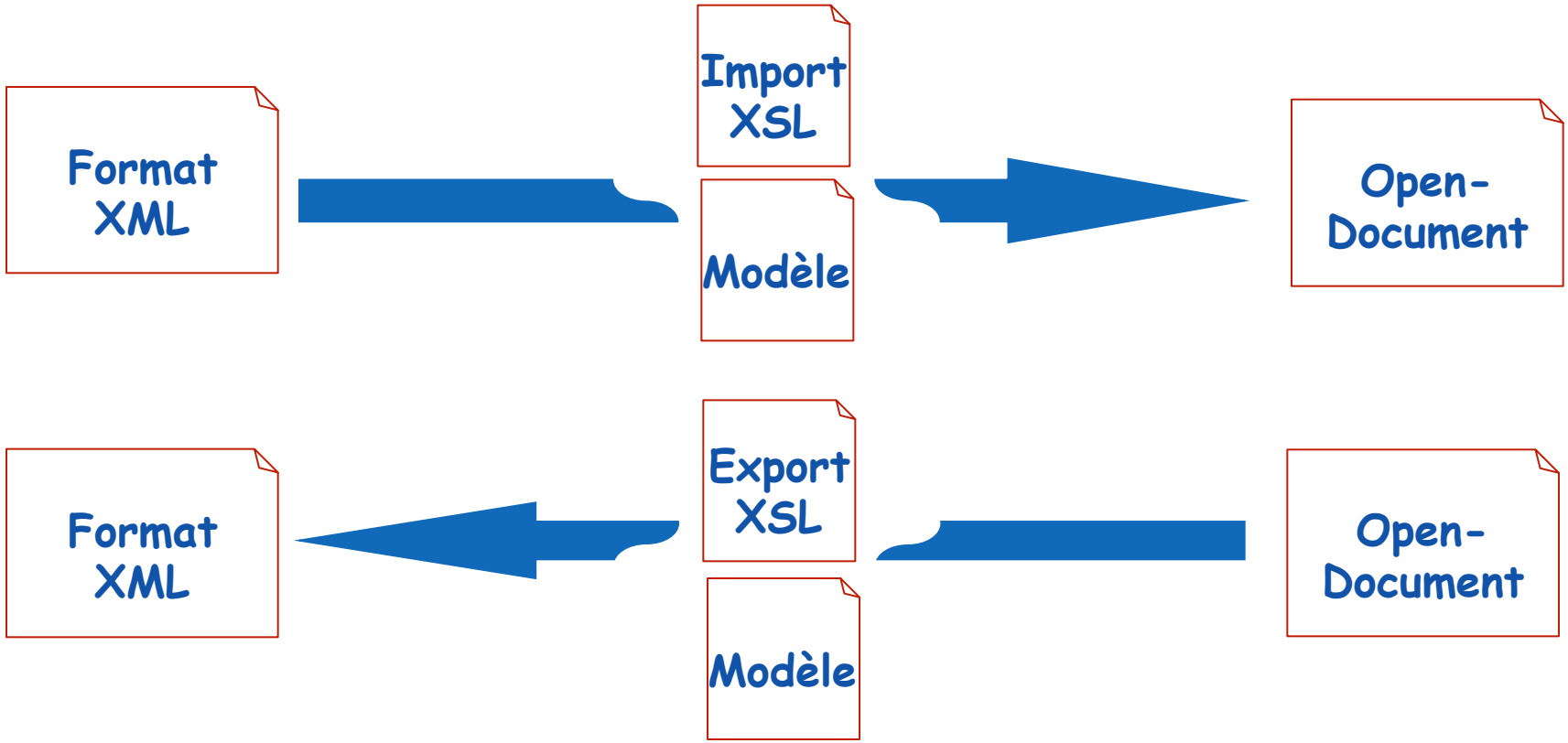
- SGBD Natif XML
  - conçu pour XML,
  - stockant les documents en entiers sans les décomposer en éléments,
  - utilisant des techniques d'indexation d'arbres spécifiques.



XML et Bureautique

- Stockage
  - Format des fichiers : XML devient possible
  - Compatibilité avec l'existant
- Feuilles de styles
  - Modèle d'import et d'export
- Des jargons spéciaux
  - Word ML, SpreadsheetML, DataML
  - Open Document 1.0 de l'Oasis

XML et Bureautique (2)



Principes de la sécurité

- **Identification/Authentification**
  Qui êtes-vous, pouvez-vous le prouver ?
- **Autorisation**
  Pouvez-vous faire cette opération sur cet objet ?
- **Intégrité**
  Les données sont protégées contre toute modification (accidentelle ou malveillante)
- **Confidentialité**
  Les données restent privées et elles ne peuvent pas être vues par des utilisateurs non autorisés

Principes de la sécurité (2)

- Audit
  un audit et une journalisation sont essentiels pour résoudre les problèmes de sécurité a posteriori

- Non-répudiation
  le système peut prouver qu'un utilisateur a fait une opération

- Disponibilité
  capacité pour des systèmes de rester disponibles pour les utilisateurs légitimes (ex.: pas de refus de service)

Un peu de jargon ...

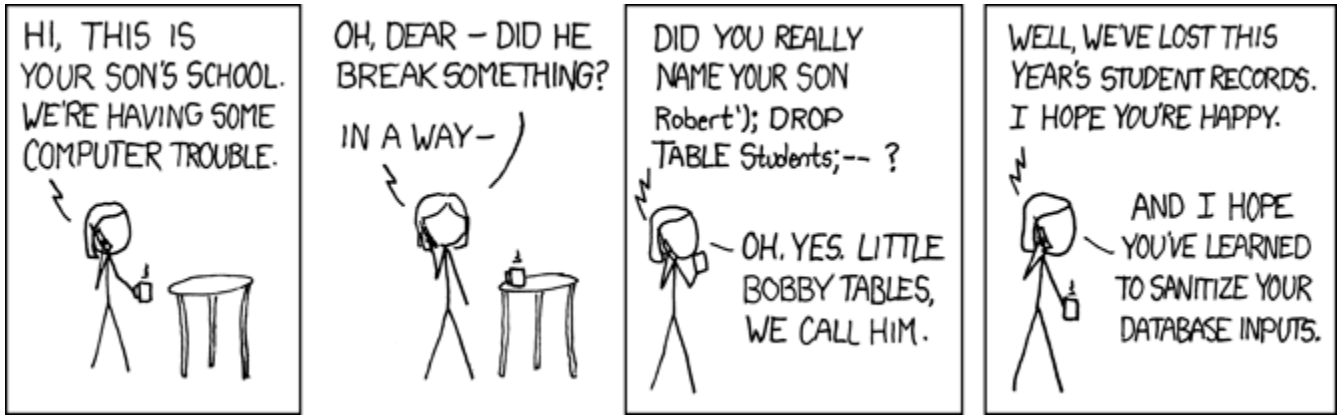
- Menace
  - C'est un événement potentiel, malveillant ou pas, qui pourrait nuire à une ressource
  - toute opération préjudiciable à vos ressources est une menace
- Vulnérabilité
  - C'est une faiblesse qui rend possible une menace
  - Elle peut être due à une mauvaise conception
  - Elle peut être due à des erreurs de configuration
  - Elle peut être due à des techniques de codage inappropriées et non fiables

Un peu de jargon ... (2)

- Attaque
  - C'est une action qui exploite une vulnérabilité ou exécute une menace
  - Quelques exemples
    - Envoyer des données d'entrée malveillantes à une application
    - Saturer un réseau en vue d'entraîner un refus de service

Exemple : Injection du XML

- Les applications web modernes utilisent des formulaires
- L'utilisateur remplit les champs du formulaire
- Une application traite ces informations
- Évidemment, l'utilisateur est honnête, c'est sûr !



Exemple : Injection du XML (2)

- Sauf que …
  - le monde n'est pas parfait

- Les programmes peuvent recevoir n'importe quoi
  - Beaucoup d'applications web utilisent désormais une base de données
  - les requêtes sur la base utilisent des informations issues de l'internaute
  - il faut valider ces informations avant de les utiliser, sinon…

Exemple : Injection du XML (3)

- Sinon, risque d'injection !
- C'est quoi ?
  - concaténer une chaîne de caractères pour créer du code XML
- Pourquoi faire ?
  - contourner des règles de gestion : obtenir des informations auxquelles on ne devrait pas avoir accès
  - altérer des données de la base de données : modifier ou supprimer des données
- Circonstance aggravante : afficher les erreurs du parser

Exemple : Injection du XML (4)

- Supposons que l'on a une application Web pour gérer des données stockées dans une BD XML.

- L'application utilise une communication de style XML afin d'effectuer l'enregistrement de nouveaux utilisateurs.

- Cela se fait par la création et l'ajout d'un nouveau nœud <utilisateur> dans un fichier XML.

Exemple : Injection du XML (5)

- Voici le fichier XML

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<users>
    <user>
        <username>gandalf</username>
        <password>!c3</password>
        <userid>0</userid>
        <mail>gandalf@middleearth.com</mail>
    </user>
    <user>
        <username>Stefan0</username>
        <password>w1s3c</password>
        <userid>500</userid>
        <mail>Stefan0@whysec.hmm</mail>
    </user>
</users>
```

Exemple : Injection du XML (6)

- Un utilisateur s'inscrit lui-même en introduisant les valeurs suivantes :

Username : tony
Password : Un6R34kb!e
E-mail : s4tan@hell.com

- Le message envoyé est :

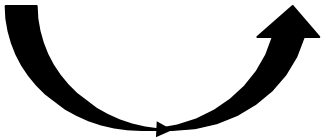
http://www.example.com/addUser.php?
username=tony&password=Un6R34kb!e&email=s4tan@hell.com

Exemple : Injection du XML (7)

- L'application ajoute un nouveau noeud

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<users>
    <user>
        <username>gandalf</username>
        <password>!c3</password>
        <userid>0</userid>
        <mail>gandalf@middleearth.com</mail>
    </user>
    <user>
        <username>Stefan0</username>
        <password>w1s3c</password>
        <userid>500</userid>
        <mail>Stefan0@whysec.hmm</mail>
    </user>

    <user>
        <username>tony</username>
        <password>Un6R34kb!e</password>
        <userid>500</userid>
        <mail>s4tan@hell.com</mail>
    </user>
</users>
```



Exemple : Injection du XML (8)

- Supposons que notre utilisateur tape les valeurs suivantes

Username : tony
Password : Un6R34kb!e
E-mail : s4tan@hell.com</mail><userid>0</userid><mail>s4tan@hell.com

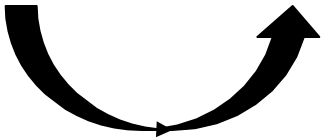
Exemple : Injection du XML (9)

- L'application obtiendra le fichier XML suivant :

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<users>
    <user>
        <username>gandalf</username>
        <password>!c3</password>
        <userid>0</userid>
        <mail>gandalf@middleearth.com</mail>
    </user>
    <user>
        <username>Stefan0</username>
        <password>w1s3c</password>
        <userid>500</userid>
        <mail>Stefan0@whysec.hmm</mail>
    </user>

    <user>
        <username>tony</username>
        <password>Un6R34kb!e</password>
        <userid>500</userid>
        <mail> s4tan@hell.com</
mail><userid>0</userid><mail>s4tan@hell.com
        </mail>
    </user>

</users>
```



Exemple : Injection du XML (10)

- Supposons que notre fichier XML est spécifié selon la DTD suivante :

```
<!DOCTYPE users [
    <!ELEMENT users (user+) >
    <!ELEMENT user (username,password,userid,mail+) >
    <!ELEMENT username (#PCDATA) >
    <!ELEMENT password (#PCDATA) >
    <!ELEMENT userid (#PCDATA) >
    <!ELEMENT mail (#PCDATA) >
]>
```

Exemple : Injection du XML (11)

- Pour avoir un document XML valide à sa DTD, notre utilisateur tape les valeurs suivantes

Username : tony
Password : Un6R34kb!e</password><!--
E-mail : --><userid>0</userid><mail>s4tan@hell.com

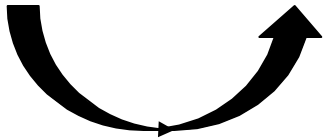
Exemple : Injection du XML (12)

- L'application obtiendra le fichier XML suivant :

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<users>
    <user>
        <username>gandalf</username>
        <password>!c3</password>
        <userid>0</userid>
        <mail>gandalf@middleearth.com</mail>
    </user>
    <user>
        <username>Stefan0</username>
        <password>w1s3c</password>
        <userid>500</userid>
        <mail>Stefan0@whysec.hmm</mail>
    </user>

    <user>
        <username>tony</username>
        <password>Un6R34kb!e</password><!--
        </password>
        <userid>500</userid>
        <mail>-->
        <userid>0</userid><mail>s4tan@hell.com</mail>
    </user>

</users>
```



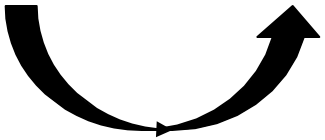
Exemple : Injection du XML (13)

- L'application obtiendra le fichier XML suivant :

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<users>
    <user>
        <username>gandalf</username>
        <password>!c3</password>
        <userid>0</userid>
        <mail>gandalf@middleearth.com</mail>
    </user>
    <user>
        <username>Stefan0</username>
        <password>w1s3c</password>
        <userid>500</userid>
        <mail>Stefan0@whysec.hmm</mail>
    </user>

    <user>
        <username>tony</username>
        <password>Un6R34kb!e
        </password>
<!--  </password><userid>500</userid><mail>-->
        <userid>0</userid>
        <mail>s4tan@hell.com</mail>
    </user>

</users>
```



Sommaire

- Introduction
- Signature de XML
- Attaques contre XML

Définition

- Aussi appelée signature électronique, la signature numérique doit remplir les mêmes fonctions que son homologue manuscrit, à savoir :

  - Garantir l'authenticité de l'expéditeur d'un message
  - Vérifier l'intégrité du message transmis
  - Assurer la non répudiation d'un envoi

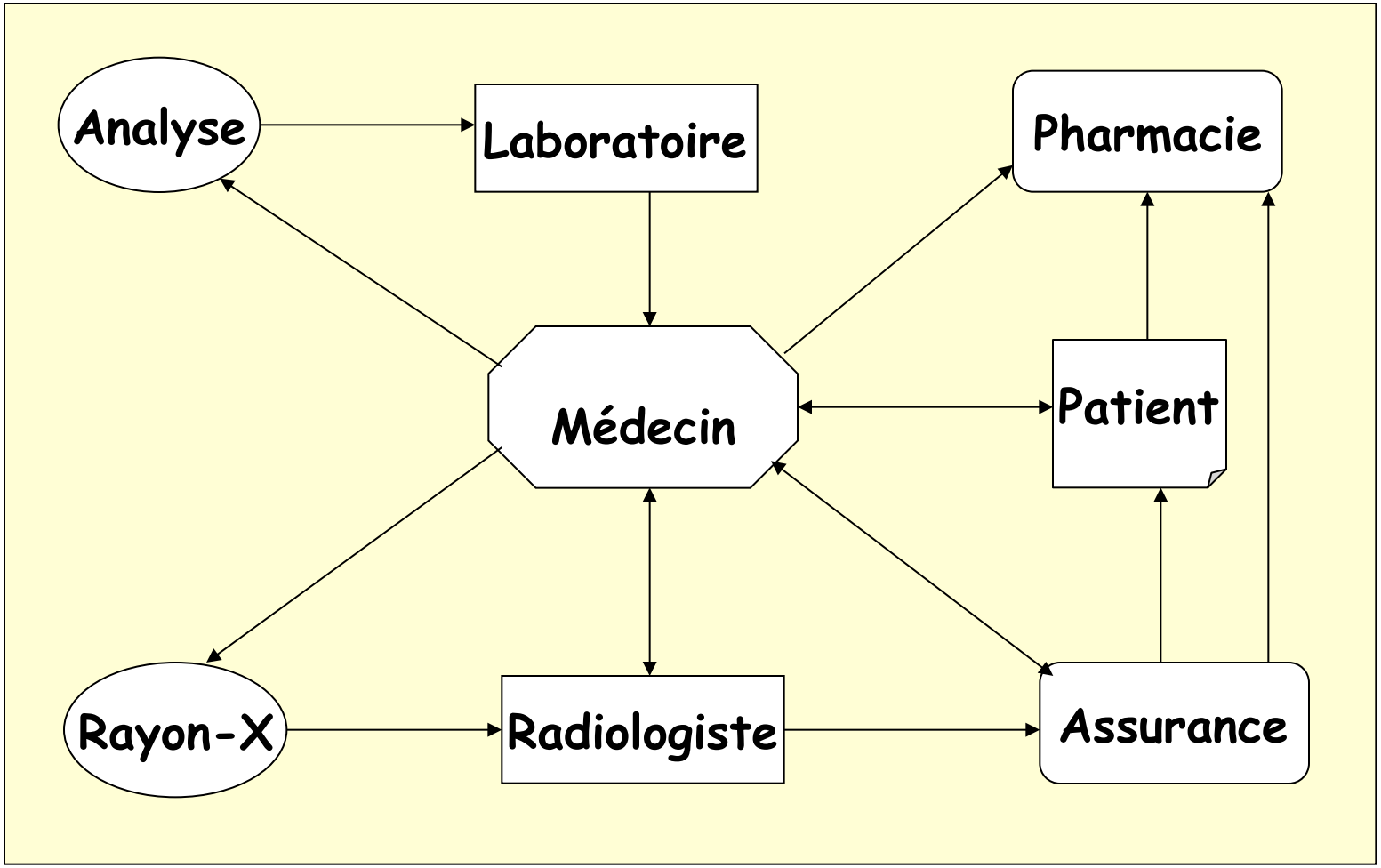
Objectifs

- Fournir un cadre normalisé pour la signature numérique, entre autres des documents XML, mais pas exclusivement
- Préoccupation d'interopérabilté des signatures numériques
- Supporter les techniques de cryptographie (clés secrètes)

Exemple

- Le dossier du patient est maintenu pendant une longue période du temps
- Le dossier peut être consulté et modifié par plusieurs médecins
- Chaque médecin n'est responsable que d'une partie du dossier

Exemple (2)


Analyse
Laboratoire
Pharmacie
Médecin
Patient
Rayon-X
Radiologiste
Assurance

Exemple (3)

- Informations du patient

```
<Patient name="Fernandel" id="1234">
   <Address>…</Address>
   <PreviousVisits>…</PreviousVisits>
</Patient>
```

- Le médecin examine le patient et lui recommande de faire une analyse de sang ainsi qu'une radio …

```
<Patient name="Fernandel" id="1234">
…
   <Visits date="10/01/2009"  id="10-10-Physical">
        <DoctorCheckUp DoctorName="Bourvil">
              <BloodPressure>…</BloodPressure>
              <PatientComplaint>Douleur au genou droit</PatientComplaint>
              <Comments>Analyse de sang et radio recommandées</Comments>
 </DoctorCheckUp></Visits></Patient>
```

Exemple (4)

- Laboratoire enregistre les résultats de l'analyse

```
<Visits date="10/01/2009"  id="10-10-Physical">
…
   <Lab date="12/01/2009" >
     <LabTechnician name="Louis" id="AB2356">
                <Blood LDL="123" HDL="456" Total="217/>
     </Lab>
</Visits>
```

- Laboratoire de radio. Attache l'image de la radio

```
<Visits date="10/01/2009"  id="10-10-Physical">
…
   <Xray date="12/01/2009" >
     <LabTechnician name="Robert" id="ZZ5611">
                <Image type="GenouDroit"
              ref="http://xlab.com/Fernandel/12/01/09/GenouD.jpeg"/>
     </Xray>
</Visits>
```

Exemple (5)

- Le médecin donne ses dernières recommandations

```
<Visits date="10/01/2009"  id="10-10-Physical">
…
   <Recommendations DoctorName="Bourvil" >
      <Comments> Recommande régime et médicaments.</Comments>
      <Prescription> <Tablet dose="une fois par jour">xyz</Tablet>
      </Prescription>
   </Recommendations></Visits>
```

- Le médecin envoie l'ordonnance à la pharmacie

```
<Visits date="10/01/2009"  id="10-10-Physical">
…
   <Recommendations DoctorName="Bourvil" >
      <Comments> Recommande régime et médicaments.</Comments>
      <Prescription> <Tablet dose="une fois par jour">xyz</Tablet>
      </Prescription>
   </Recommendations></Visits>
```

Prérquis

- La signature numérique est basée sur la cryptographie à clé publique, dans laquelle chaque individu d'un groupe possède une paire de clés :
  - une clé secrète (privée) connue seulement de l'individu
  - une clé publique, connue de tout le groupe

Prérquis (2)

Cryptage avec une clé privée

- En utilisant un algorithme standard (connu de tous) chaque individu peut crypter un message avec sa clé privée
- Exemple :
  - Individu: *A*
  - Sa clé secrète: S_A
  - Message: *M*
  - Message crypté avec la clé privée de *A*: $S_A(M)$

Prérquis (3)

Décryptage avec une clé publique

- En utilisant un algorithme standard (connu de tous) chaque individu peut décrypter un message crypté en utilisant une clé prublique

- Le décryptage ne va donner quelque chose de sensée que si la clé publique utilisée correspond à la clé privée utilisée pour le cryptage

Prérquis (4)

Un premier exemple

- Individu qui crypte : A
- Clé privée de A : $\mathcal{S}_A$
- Message crypté : $\mathcal{S}_A(M) = C$
- Clé publique de A : $\mathcal{P}_A$
- Décryptage avec $\mathcal{P}_A$: $\mathcal{P}_A(C) = \mathcal{P}_A(\mathcal{S}_A(M)) = M$
- On obtient le message originel M

Prérquis (4)

Un deuxième exemple

- Individu qui crypte: A
- Clé secrète de A: $\mathcal{S}_A$
- Message crypté: $\mathcal{S}_A(M) = C$
- Clé publique de X (autre que A): $\mathcal{P}_X$
- Décryptage avec la clé publique de X:
  $\mathcal{P}_X(C) = \mathcal{P}_X(\mathcal{S}_A(M)) = M'$, différent de M

Prérquis (5)

Une paire de clés

- Au moment où un individu se joint au groupe, les deux clés de sa paire sont générées ensemble, selon une méthode très précise (mais bien connue), de sorte que :

  il existe une relation mathématique particulière très forte entre la clé privée et la clé publique d'une paire

Prérquis (5)

Gestion des paires de clés

- Première possiblité
  Un responsable unique, jugé digne de confiance par tout le groupe, génère les paires de clés de tout le monde et fait une distribution adéquate
- Problème
  Le responsable connaît toutes les clés privées et peut (de son plein gré ou sous une menace quelconque) signer n'importe quoi au nom de n'importe qui !

Prérquis (6)

Gestion des paires de clés

- Deuxième possiblité
  - Chaque individu génère sa paire de clés sur son propre ordinateur et la clé privée ne sort jamais de cet ordinateur
  - Chaque membre du groupe fait connaître sa clé publique et prouve son identité aux autres membres à l'occasion de rencontres en personne (Internet n'étant pas assez sécuritaire)

Prérquis (7)

Gestion des paires de clés

- Une preuve d'identité suffisante doit être fournie par un membre à un autre
- Cette preuve doit être notée pour référence en même temps que la clé publique
- Dans un groupe de *n* individus, chaque personne doit effectuer *n*-1 rencontres (pour une vérifiabilité universelle)

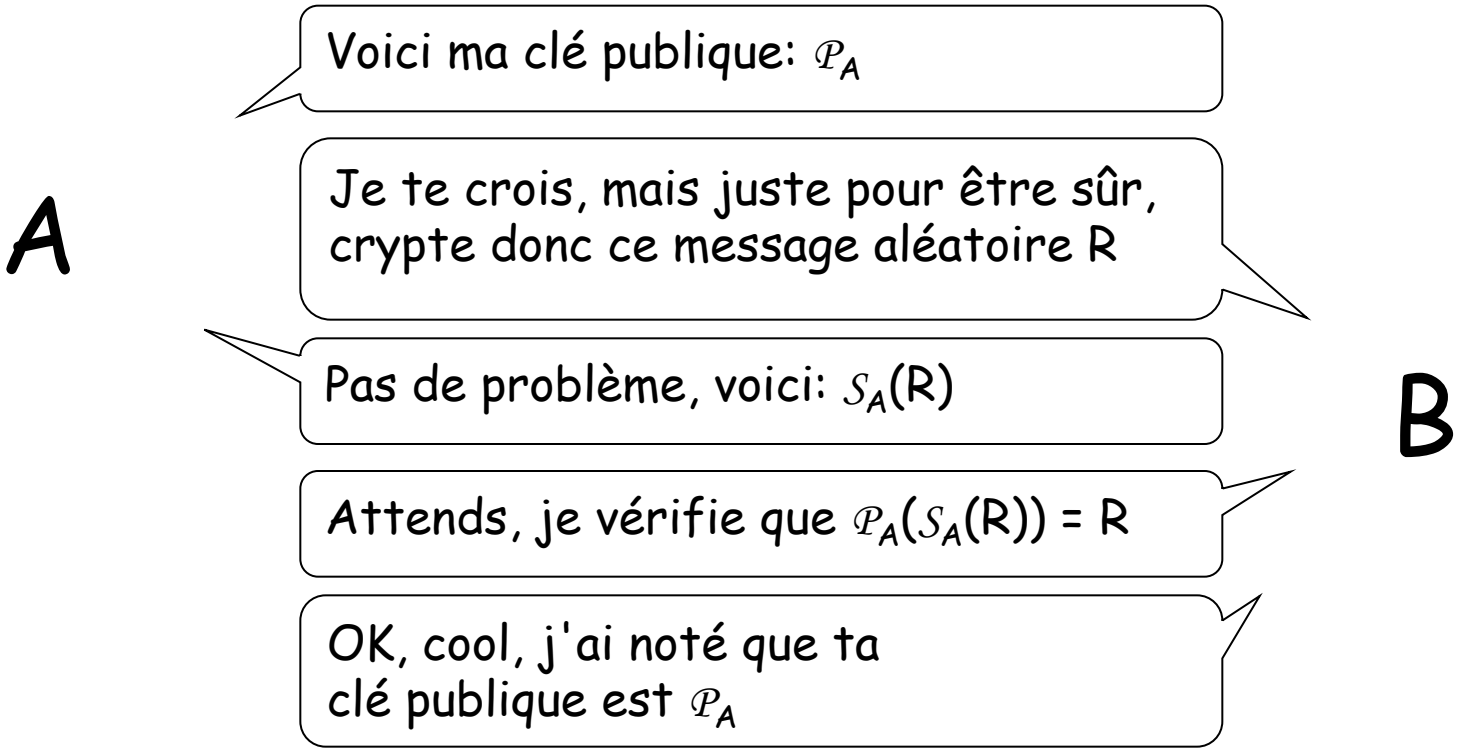
Prérquis (8)

Gestion des paires de clés

- Au moment où un membre A communique sa clé publique à un autre membre B, B vérifie que A possède bien la clé secrète correspondante en lui demandant de crypter (sans consulter personne) un message aléatoire
- A défaut de ce test, A pourrait refiler à B une clé publique obtenue d'un autre membre

Prérquis (9)

Gestion des paires de clés



Condensation du contenu

- Inefficace de crypter le message au long
- On calcule puis on crypte un condensé mathématique plus court (digest) du message
- Fonction de condensation: la moindre modification à un message donne un condensé différent (avec grande probabilité)

Condensation du contenu (2)

Fonction de hachage

- Résultat d'une fonction appelée fonction de hachage
- Conversion d'un contenu d'informations en un plus petit
- La fonction hachage doit possèder les propriétés suivantes :
  - $H(x) \neq H(y)$ implique $x \neq y$
  - $H(x) = H(y)$ implique $x = y$

Condensation du contenu (3)

- Intérêts
  - Signature sur des données plus petites
  - Vérification des données signées
  - Non réversible
- Il existe plusieurs algorithmes différents pour le cryptage et la condensation
- Pour l'interopérabilité, la signature doit faire mention des algorithmes utilisés
- Plus connus:
  - Cryptage: DSA, RSA (1978, Turing Award 2002)
  - Condensation: SHA-1, MD5 (déconseillé)

Canonisation du contenu

- Deux éléments XML équivalents mais différents
  - <elem id = ‘0’ xml:lang = ‘fr’>
  - <elem xml:lang = ‘fr’ id = ‘0’ >
- Condensats différents
  - dd00bd0a32e382e508112ac869f2ddad
  - D0a3f919ca76b518705fc461331fb5a0
- Signatures différentes

Canonisation du contenu (2)

Principe

- Procédé par lequel on convertit des données qui ont plusieurs représentations possibles vers un format 'standard'
- Pour XML, la canonisation procède (entre autres) par les transformations suivantes :
  - Suppression des espaces inutiles dans les balises
  - Utilisation d'un codage particulier pour les caractères
  - Tri des attributs selon un ordre lexicographique

Idée de signature

- Supposons que l'individu *A* envoie une message *M* signé à tous les membres d'un groupe

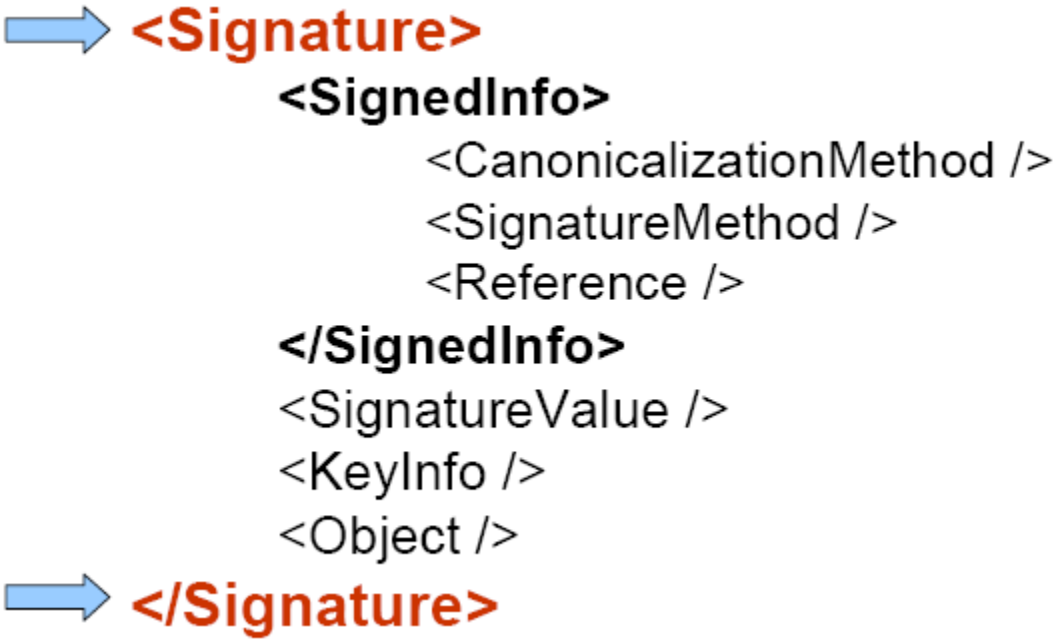
- Pour ce faire *A* doit procéder comme suit :
  - Condenser M : $M' = H(M)$
  - Crypter le condensat : $S_A(M')$
  - Transmettre son message en double : une fois en clair, une fois crypté avec sa clé privée

Idée de signature (2)

- Tout le monde connaît la clé publique ($\mathcal{P}_A$) de l'auteur prétendu (**A**) et peut l'utiliser pour décrypter le message crypté $\mathcal{S}_A$(**M**)
- Si le message en clair **M** et le message décrypté $\mathcal{P}_A(\mathcal{S}_A$(**M**)) sont identiques, cela prouve que c'est bien la clé privée de l'auteur prétendu (**A**) qui a été utilisée pour générer le message crypté

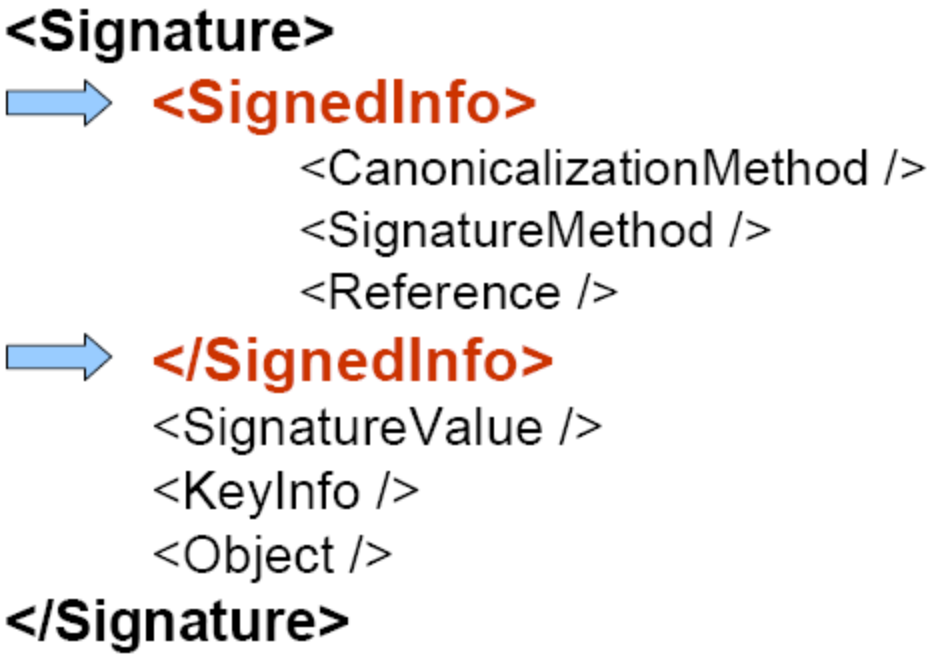
Syntaxe

- W3C et IETF
  XML Digital Signature http://www.w3.org/TR/xmldsig-core/



Syntaxe (2)

SignedInfo contient l'information à signer



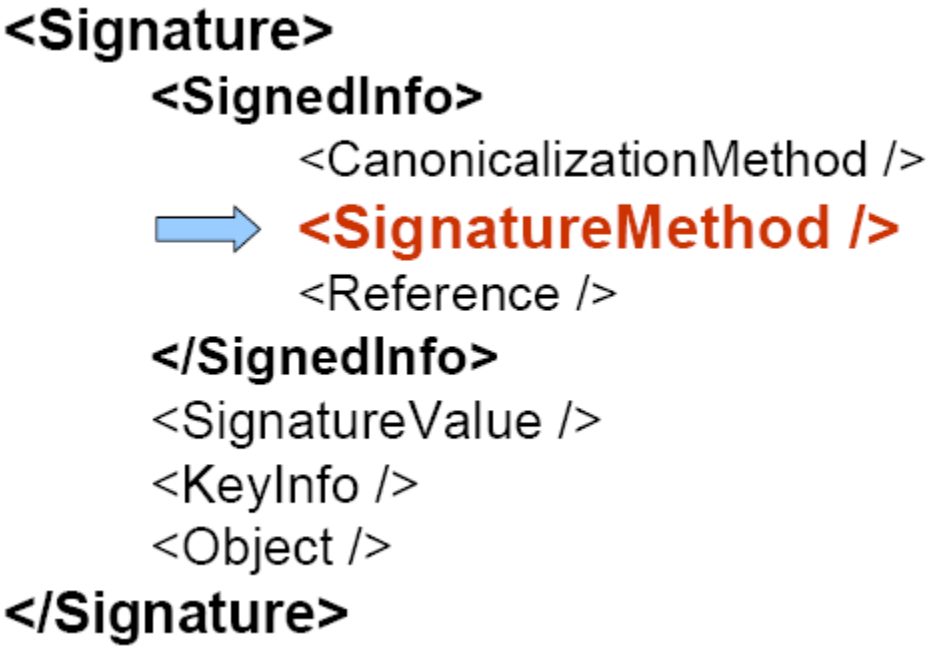
Syntaxe (3)

CanonicalizationMethod contient l'algorithme de canonisation

```
<Signature>
    <SignedInfo>
        <CanonicalizationMethod />
        <SignatureMethod />
        <Reference />
    </SignedInfo>
    <SignatureValue />
    <KeyInfo />
    <Object />
</Signature>
```

Syntaxe (4)

SignetureMethod contient l'algorithme de signature



Syntaxe (5)

Reference identifie la ressource à signer

```
<Signature>
    <SignedInfo>
        <CanonicalizationMethod />
        <SignatureMethod />
        <Reference />
    </SignedInfo>
    <SignatureValue />
    <KeyInfo />
    <Object />
</Signature>
```

Syntaxe (6)

Transforms contient les transformations à appliquer avant la signature

```
<Signature>
    <SignedInfo>
        <Reference>
            <Transforms />
            <DigestMethod />
            <DigestValue />
        </Reference>
    </SignedInfo>
    <SignatureValue />
    <KeyInfo />
    <Object />
</Signature>
```

Syntaxe (7)

DigestMethod contient l'algorithme de condensation

```
<Signature>
    <SignedInfo>
        <Reference>
            <Transforms />
            <DigestMethod />
            <DigestValue />
        </Reference>
    </SignedInfo>
    <SignatureValue />
    <KeyInfo />
    <Object />
</Signature>
```

Syntaxe (8)

DigestValue contient le résulat de l'algorithme de condensation (digest)

```
<Signature>
    <SignedInfo>
        <Reference>
            <Transforms />
            <DigestMethod />
            <DigestValue />
        </Reference>
    </SignedInfo>
    <SignatureValue />
    <KeyInfo />
    <Object />
</Signature>
```

Syntaxe (9)

SignatureValue contient le résulat de l'algorithme de signature

```
<Signature>
    <SignedInfo>
        <CanonicalizationMethod />
        <SignatureMethod />
        <Reference />
    </SignedInfo>
    <SignatureValue />
    <KeyInfo />
    <Object />
</Signature>
```

Syntaxe (10)

KeyInfo contient la clé nécessaire pour valider le document

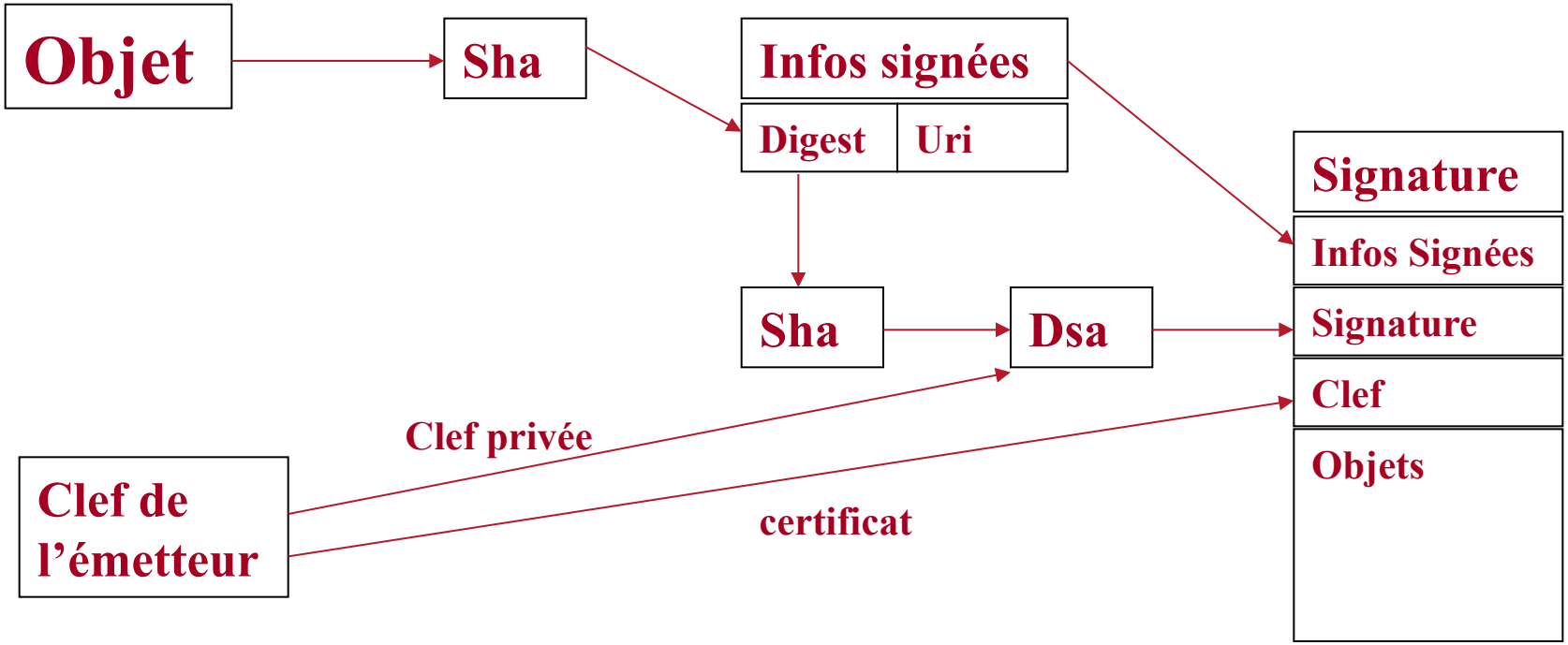
```
<Signature>
    <SignedInfo>
        <CanonicalizationMethod />
        <SignatureMethod />
        <Reference />
    </SignedInfo>
    <SignatureValue />
    <KeyInfo />
    <Object />
</Signature>
```

Syntaxe (11)

Object contient la donnée que nous souhaitons signer (option)

```
<Signature>
    <SignedInfo>
        <CanonicalizationMethod />
        <SignatureMethod />
        <Reference />
    </SignedInfo>
    <SignatureValue />
    <KeyInfo />
    <Object />
</Signature>
```

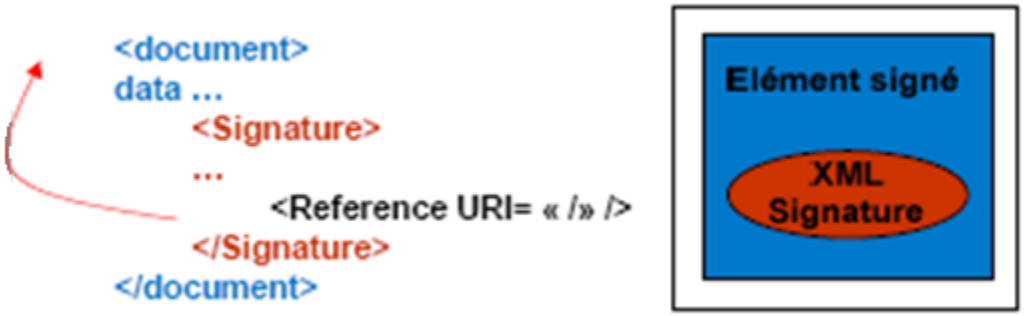
Construction du message



Formes de signature

1) Signature enveloppée (enveloped signature)

- Lorsque la signature s'applique aux données qui l'entoure dans le reste du document



Formes de signature (2)

Exemple de Signature enveloppée

```
<?xml version="1.0"?>
<Envelope>
  <Data>contenu</Data>
<Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
  <SignedInfo>
    <CanonicalizationMethod
                    Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
    <SignatureMethod
                   Algorithm="http://www.w3.org/2000/09/xmldsig#hmac-sha1"/>
    <Reference>
      <Transforms>
        <Transform
            Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature"/>
      </Transforms>
      <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
      <DigestValue>MMMkB0ZPp82XrUvJMFqDIEuXy0o=</DigestValue>
    </Reference>
  </SignedInfo>
  <SignatureValue>mVPvfcVSXi9elKL+IcSCAzD4Jbk=</SignatureValue>
</Signature>
</Envelope>
```

Formes de signature (3)

2) Signature enveloppante (enveloping signature)

- Lorsque les données signées forment un sous élément de la signature elle-même

```
<Signature>
...
        <Reference URI= «  #data » />
        <Object id =«data» >
                <document>
                data ...
                </document>
        </Object>
...
</Signature>
```

Formes de signature (4)

Exemple de Signature enveloppante

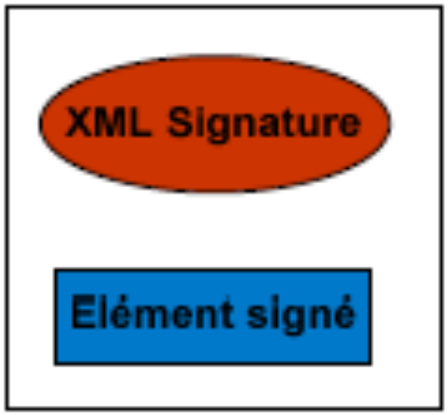
```
<?xml version="1.0"?>
<Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
  <SignedInfo>
    <CanonicalizationMethod
                Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
    <SignatureMethod
                Algorithm="http://www.w3.org/2000/09/xmldsig#hmac-sha1"/>
    <Reference URI="#myobj">
      <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
      <DigestValue>C2g9BLcGyGPCVKuF2byR1Ym+6pE=</DigestValue>
    </Reference>
  </SignedInfo>
   <SignatureValue>+R/XEOHDvR/jbmmpiuH4ZcRqC6c=</SignatureValue>
  <Object Id="myobj">Salut tout le monde!</Object>
</Signature>
```

Formes de signature (5)

2) Signature détachée (detached signature)

- Lorsque la signature concerne des ressources extérieures au document qui la contient

```
<Signature>
...
      <Reference
      URI= « http://ex.com/doc.txt » />
...
</Signature>
```



Formes de signature (6)

Exemple de Signature détachée

```
<?xml version="1.0"?>
<Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
  <SignedInfo>
    <CanonicalizationMethod
                Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
    <SignatureMethod
                Algorithm="http://www.w3.org/2000/09/xmldsig#hmac-sha1"/>
    <Reference URI="http://www.loria.fr/text.txt">
      <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
      <DigestValue>oLZZOWcLwsAQ9NXWoLPk5FkPuSs=</DigestValue>
    </Reference>
  </SignedInfo>
<SignatureValue>O9ykpFMXmkddzJ3CySrpzHBUW/Q=</SignatureValue>
</Signature>
```

Formes de signature (7)

Un autre exemple : Hello World!

```
<?xml version="1.0"?>
<Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
  <SignedInfo>
    <CanonicalizationMethod
                Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
    <SignatureMethod
               Algorithm="http://www.w3.org/2000/09/xmldsig#hmac-sha1"/>
    <Reference URI="#myobj">
      <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
      <DigestValue>C2g9BLcGyGPCVKuF2byR1Ym+6pE=</DigestValue>
    </Reference>
  </SignedInfo>
   <SignatureValue>+R/XEOHDvR/jbmmpiuH4ZcRqC6c=</SignatureValue>
  <Object Id="myobj">Hello World!</Object>
</Signature>
```

Algorithmes de canonisation

- Rendre la signature indépendante des différentes représentations internes d'un document XML, de façon à obtenir la vérifiabilité universelle
- Deux version standards
  - Sans commentaires

    http://www.w3.org/TR/2001/REC-xml-c14n-20010315
  - Avec commentaires

    http://www.w3.org/TR/2001/REC-xml-c14n-20010315#WithComments

Algorithmes de transformation

- Procéder à des transformations arbitraires des données avant la condensation
- Trois transformations pour XML:
  - Enveloped Signature
    http://www.w3.org/2000/09/xmldsig#enveloped-signature
  - Xpath
    http://www.w3.org/TR/1999/REC-xpath-19991116
  - XSLT
    http://www.w3.org/TR/1999/REC-xslt-19991116

Algorithmes de transformation (2)

- Exemple d'utilisation de Xpath

```
<Reference URI="file:///home/imine/xml/article.xml">
<Transforms>
<Transform Algorithm="http://www.w3.org/TR/1999/REC-xpath-1999116">
      <XPath>
            //article[@Id="Imine"]
      </XPath>
</Transform>
</Transforms>
</Reference>
```

Protocole de communication

- Emetteur
  - Création du message
  - Canonisation
  - Signature
- Récepteur
  - Réception du message
  - Canonisation
  - Vérification de la signature

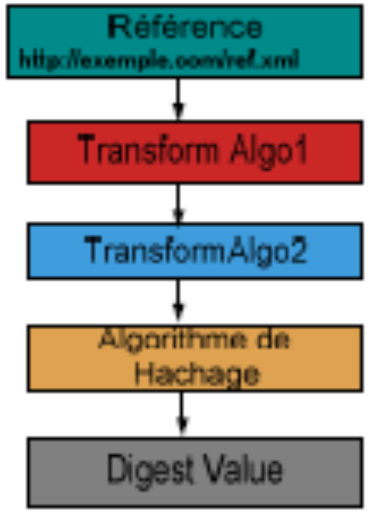
Génération de la signature

1. Appliquer les transformations sur les données
2. Calculer la valeur du condensat (DigestValue)
3. Créer l'élément Reference
4. Créer l'élément SignedInfo (avec intégration des éléments SignatureMethod, CanonicalizatioMethod, et Reference
5. Canoniser l'élément SignedInfo
6. Calculer la valeur de la signature (SignatureValue) sur la forme canonisée de SignedInfo et en se basant sur la méthode de signature indiquée (SignatureMethod)
7. Créer l'élément Signature

Génération de la signature (2)

Génération de la référence

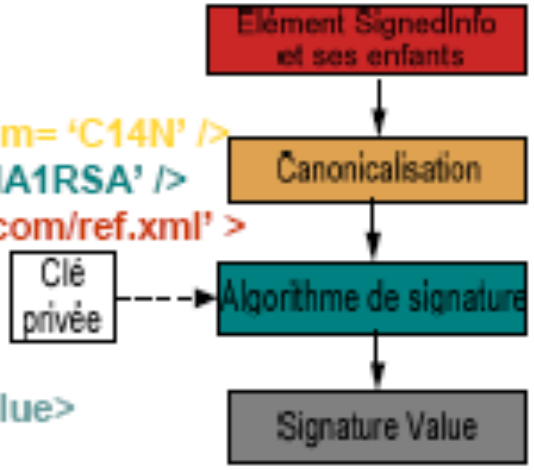
```
<Reference URI='http://exemple.com/ref.xml'>
  <Transforms>
    <Transform Algorithm= 'Algo1' />
    <Transform Algorithm= 'Algo2' />
  </Transforms>
  <DigestMethod Algorithm = 'SHA1' />
  <DigestValue>...<DigestValue />
</Reference>
```



Génération de la signature (2)

Génération de la signature

```
<Signature>
  <SignedInfo>
    <CanonicalizationMethod Algorithm= 'C14N' />
    <SignatureMethod Algorithm= 'SHA1RSA' />
    <Reference URI = 'http://exemple.com/ref.xml' >
     .....
     </Reference>
  </SignedInfo>
  <SignatureValue /> ... </SignatureValue>
<Signature>
```



Validation de la signature

1. Pour chaque élément Reference de SignedInfo :
   a. Obtenir la donnée à condenser
   b. Condenser la donnée en utilisant la méthode indiquée dans DigestMethod
   c. Comparer la valeur calculée avec celle enregistrée dans DigestValue. S'il y a divergence, alors la validation échoue.
2. Obtenir la clé publique à partir de l'élément keyInfo ou d'une source externe
3. Canoniser l'élément SignedInfo selon l'algorithme indiqué dans l'élément CanonicalizatioMethod
4. Confirmer la valeur stockée dans SignatureValue sur l'élément SignedInfo

Validation de la signature (2)

Validation de la référence

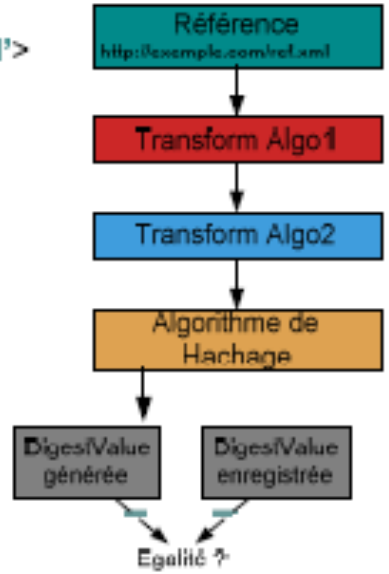
```
<Reference URI= 'http://exemple.com/ref.xml'>

  <Transforms>
    <Transform Algorithm= 'Algo1' />

    <Transform Algorithm= 'Algo2' />
  </Transforms>

  <DigestMethod Algorithm = 'SHA1' />

  <DigestValue> .... </DigestValue>
</Reference>
```



Validation de la signature (3)

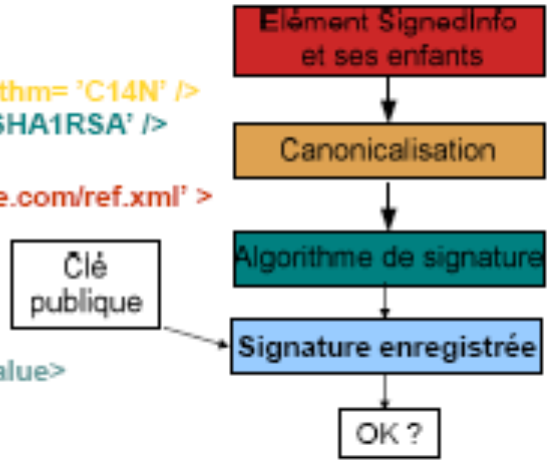
Validation de la signature

```
<Signature>
   <SignedInfo>

      <CanonicalizationMethod Algorithm= 'C14N' />
      <SignatureMethod Algorithm= 'SHA1RSA' />

      <Reference URI = 'http://exemple.com/ref.xml' >
       .....
       </Reference>

   </SignedInfo>
   <SignatureValue> ... </SignatureValue>
<Signature>
```



Implantations

- Libres
  - JSR 105 API (Java XML Digital Signature)
    http://java.sun.com/javase/6/docs/technotes/guides/security/xmldsig/XMLDigitalSignature.html
  - Apache XML Security Java
  - IBM alphaWorks XML Security Suite
  - C XML Security Library
- Commerciales
  - Baltimore Technologies KeyTools
  - Webgetail Communications Java Crypto and Security Implementation
  - Microsoft SDK .Net

API Java

- javax.xml.crypto
- javax.xml.crypto.dom
- javax.xml.crypto.dsig
- javax.xml.crypto.dsig.dom
- javax.xml.crypto.dsig.keyinfo
- javax.xml.crypto.dsig.spec

Exemple

```
// création des clés
KeyPairGenerator kpg = KeyPairGenerator.getInstance("DSA");
kpg.initialize(512);
KeyPair kp = kpg.generateKeyPair();

// signature factory
XMLSignatureFactory signatureFactory =
XMLSignatureFactory.getInstance("DOM");
DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
dbf.setNamespaceAware(true);

// lecture du fichier xml en entrée
DocumentBuilder builder = dbf.newDocumentBuilder();
Document document = builder.parse(new FileInputStream(path));
```

Exemple (1)

```
// creation de la reference
 Reference reference = signatureFactory.newReference("",
signatureFactory.newDigestMethod(DigestMethod.SHA1, null),
transforms,null, null);

// creation de signedInfo
SignedInfo si = signatureFactory.newSignedInfo
(signatureFactory.newCanonicalizationMethod
(CanonicalizationMethod.INCLUSIVE_WITH_COMMENTS,
(C14NMethodParameterSpec) null),
signatureFactory.newSignatureMethod(SignatureMethod.DSA_SHA1, null),
Collections.singletonList(reference));
```

Exemple (2)

```
// création de KeyInfo
KeyInfoFactory kif = signatureFactory.getKeyInfoFactory();
KeyValue kv = kif.newKeyValue(kp.getPublic());
KeyInfo ki = kif.newKeyInfo(Collections.singletonList(kv));

// création de la signature
DOMSignContext dsc = new DOMSignContext(kp.getPrivate(), nodeSign);
XMLSignature signature = signatureFactory.newXMLSignature(si, ki);
      signature.sign(dsc);

// écriture du fichier resultat
FileOutputStream fos = new FileOutputStream(resultat);
TransformerFactory tf = TransformerFactory.newInstance();
Transformer trans = tf.newTransformer();
 trans.transform(new DOMSource(document), new StreamResult(fos));
```

Sommaire

- Introduction
- Signature de XML
- Attaques contre XML

C14N : Expansion d'Entités

- Objet d'attaque
  - Elle peut être déclenchée au moment de la canonisation
- Risque
  - Déni de Service
- Description
  - C14N peut être une opération coûteuse, nécessitant un traitement complexe, y compris l'expansion d'entités et la normalisation des espaces. Cela nécessite la construction d'un DOM qui peut être très couteux en termes de temps et de mémoire

C14N : Expansion d'Entités (2)

- Scénario
  - Alice envoie un document XML signé à Bob
  - L'intrus intercepte ce document et remplace l'élément SignedInfo ou le contenu XML identifié par une référence par un ensemble très large de données XML (e.g. contenant un nombre important de déclarations)
  - L'intrus envoie ensuite le document XML modifié à Bob
  - Une fois reçu par Bob, la validation de ce document conduira à un déni de service

C14N : Expansion d'Entités (3)

- Exemple

```
<!DOCTYPE foo [
<!ENTITY a "1234567890" >
<!ENTITY b "&a;&a;&a;&a;&a;&a;&a;&a;" >
<!ENTITY c "&b;&b;&b;&b;&b;&b;&b;&b;" >
<!ENTITY d "&c;&c;&c;&c;&c;&c;&c;&c;" >
<!ENTITY e "&d;&d;&d;&d;&d;&d;&d;&d;" >
<!ENTITY f "&e;&e;&e;&e;&e;&e;&e;&e;" >
<!ENTITY g "&f;&f;&f;&f;&f;&f;&f;&f;" >
<!ENTITY h "&g;&g;&g;&g;&g;&g;&g;&g;" >
<!ENTITY i "&h;&h;&h;&h;&h;&h;&h;&h;" >
<!ENTITY j "&i;&i;&i;&i;&i;&i;&i;&i;" >
<!ENTITY k "&j;&j;&j;&j;&j;&j;&j;&j;" >
<!ENTITY l "&k;&k;&k;&k;&k;&k;&k;&k;" >
<!ENTITY m "&l;&l;&l;&l;&l;&l;&l;&l;" >
]>
<foo>&m;</foo>
```

C14N : Expansion d'Entités (4)

- Quelques mesures préventives
  - Limiter la taille totale du document XML à canoniser
  - Identifier et enlever les déclarations de DTD

Injection de transformations

- Objet d'attaque
  - Elle peut être déclenchée au moment des transformations
- Risque
  - Déni de Service
  - Potentielle exécution de code arbitraire

Injection de transformations (2)

- Description
  - L'élément Transforms de Reference ou RetrievalMethod contient les instructions de traitement pour arriver à un « bon » digest en affinant la sélection des données et la transformation de ces données.
  - Un intrus peut injecter des transformations supplémentaires dans RetrievalMethod ou Reference. Ces transformations permettent de spécifier une série d'actions qui peut être utilisée pour effectuer une attaque par déni de service ou, dans certaines circonstances, même exécuter du code arbitraire.

Injection de C14N

- Risque
  - Déni de Service
- Scénario
  - Tout processeur de signatures doit mettre en œuvre une transformation C14N pour traiter du contenu XML.
  - L'intrus insère plusieurs transformations C14N pour consommer le maximum de ressources.

Injection de C14N (2)

- Exemple

```
<Reference URI="file:///home/imine/xml/article.xml">
<Transforms>
<Transform Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"> </Transform>
<Transform Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"> </Transform>
<Transform Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"> </Transform>
<Transform Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"> </Transform>
<Transform Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"> </Transform>
<Transform Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"> </Transform>
<Transform Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"> </Transform>
<Transform Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"> </Transform>
<Transform Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"> </Transform>
...
</Transforms>
</Reference>
```

Injection de C14N (3)

- Quelques mesures préventives
  - Restreindre le nombre total des transformations
  - Rejeter (par validation de schéma) tout élément Reference ou RetrievalMethod précisant multiples C14N transformations (ceci peut traiter certaines signatures qui ne sont pas malveillantes)

Injection de XPath

Les références à du contenu XML peuvent être identifiées et extraites grâce à l'utilisation des expressions XPath

- Objet d'attaque
  - Elle peut être déclenchée pendant le traitement des éléments KeyInfo ou Reference
- Risque
  - Déni de Service

Injection de XPath (2)

- Description
  - Des expressions complexes de XPath peuvent être coûteuses en terme de traitement
  - Filtres Xpath permettent des opérations telles que : l'union, l'intersection et la soustraction
  - Même avec des filtres « optimaux », des expressions XPath complexes pourraient rapidement consommer beaucoup de ressources système

Injection de XPath (3)

- Exemple

```
<Reference URI="file:///home/imine/xml/article.xml">
<Transforms>
<Transform Algorithm="http://www.w3.org/TR/1999/REC-xpath-1999116">
      <XPath>
      //chapter[count(.//para) = count(.//para[contains(.,' Strawberry' )]) ]
      </XPath>
</Transform>
</Transforms>
</Reference>
```

Injection de XPath (4)

- Quelques mesures préventives
  - Ne pas traiter KeyInfo, ou les clés identifiées par RetrievalMethod.
  - Restreindre le nombre total des transformations.
  - Refuser, par la validation du schéma, toute Reference ou RetrievalMethod précisant des expressions XPath (sauf si nécessaire)
  - Identifier le contenu par une référence complète ou par ID

Exercices sur API Java pour XML

Exercice 1. Etant donné le fichier Films.xml contenant une base de données sur les films.

1. Ecrire un parseur DOM pour afficher le metteur en scène du premier film.
2. Ecrire un parseur DOM pour créer une liste (titre + année de parution) triée par ordre alphabétique selon le titre.

Structure de base d'un parseur DOM en java :

```
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.DocumentBuilder;
import org.w3c.dom.Document;
import org.w3c.dom.NodeList;
import org.w3c.dom.Node;
import org.w3c.dom.Element;
import java.io.File;

public class ParseurDOM {

  public static void main(String args[]) {

    try {
      File fXmlFile = new File("FILMS.xml");
      DocumentBuilderFactory dbFactory = DocumentBuilderFactory.newInstance();
      DocumentBuilder dBuilder = dbFactory.newDocumentBuilder();
      Document doc = dBuilder.parse(fXmlFile);
…
    } catch (Exception e) {
      e.printStackTrace();
    }
  }
}
```

Exercice 2. Etant donné le fichier Films.xml contenant une base de données sur les films.

1. Ecrire un parseur SAX pour compter le nombre de films.
2. Ecrire un parseur SAX pour afficher les titres de films réalisés par un metteur en scène donné.

Structure de base d'un parseur SAX en java :

```
import javax.xml.parsers.SAXParser;
import javax.xml.parsers.SAXParserFactory;
import org.xml.sax.Attributes;
import org.xml.sax.SAXException;
import org.xml.sax.helpers.DefaultHandler;

public class ParseurSAX {

   public static void main(String args[]) {
    try {
      SAXParserFactory factory = SAXParserFactory.newInstance();
      SAXParser saxParser = factory.newSAXParser();
      DefaultHandler handler = new DefaultHandler() {
…
        public void startDocument() throws SAXException { … }
        public void startElement(String uri, String localName,String qName,
              Attributes attributes) throws SAXException { … }
        public void endElement(String uri, String localName,
                 String qName) throws SAXException { … }
       public void characters(char ch[], int start, int length) throws SAXException { … }
…
 };
       saxParser.parse("Films.xml", handler);

     } catch (Exception e) {
       e.printStackTrace();
}}}
```

Exercice 3. Etant donné le fichier Films.xml contenant une base de données sur les films. Ecrire un parseur DOM pour évaluer les requêtes suivantes en XPath :

1. Tous les titres de films
2. Le genre du film Alien
3. Les titres de tous les films d'horreur.

Structure de base d'un parseur DOM pour des requêtes XPath :

```
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.xpath.*;
import org.w3c.dom.Document;
import org.w3c.dom.NodeList;
import org.w3c.dom.Node;
import org.w3c.dom.Element;
import java.io.File;

public class ParseurDOM {

  public static void main(String args[]) {

    try {
      File fXmlFile = new File("mon fichier");
      DocumentBuilderFactory dbFactory = DocumentBuilderFactory.newInstance();
      DocumentBuilder dBuilder = dbFactory.newDocumentBuilder();
      Document doc = dBuilder.parse(fXmlFile);
…
     XPath xPath =  XPathFactory.newInstance().newXPath();
        String expression = "ma requete";
      NodeList mes = (NodeList) xPath.compile(expression).evaluate(doc,
XPathConstants.NODESET);
      for (int i = 0; i < mes.getLength(); i++) {
                                System.out.println(mes.item(i).getFirstChild().getNodeValue());
                              }
    } catch (Exception e) {
      e.printStackTrace();
    }
  }
}
```

IUT Nancy-Charlemagne **Applications des Bases de Données**
2014-2015

TP – Interface XML pour une BD relationnelle
(TP à réaliser en binôme)

L'objectif de ce TP est d'implanter une application en java permettant de manipuler une base de données relationnelle (stockée sur le sgbd Oracle) par l'intermédiaire de requêtes écrites dans un format XML. L'application possède les caractéristiques suivantes :

1. La connexion de l'application à la BD Oracle se fait via l'API JDBC.
2. Les requêtes SELECT, INSERT, DELETE et UPDATE seront décrites en XML :

Requête	***XML***
SELECT Listes des champs **FROM** Tables **WHERE** Condition	<SELECT> <CHAMPS> <CHAMP> … </CHAMP> <CHAMP> … </CHAMP> … </CHAMPS> <TABLES> <TABLE> … </TABLE> <TABLE> … </TABLE> … </TABLES> <CONDITION> … </CONDITION> </SELECT>
INSERT INTO Table **VALUES** (Val1, Val2, …)	<INSERT> <TABLE> … </TABLE> <VALUES> <VALUE> … </VALUE> … </VALUES> <VALUES> …. </VALUES> </INSERT>
DELETE FROM Table ***WHERE*** *Condition*	<DELETE> <TABLE> … </TABLE> <CONDITION> … </CONDITION> </DELETE>
UPDATE Table **SET** Champ = Value ***WHERE*** *Condition*	<UPDATE> <TABLE> … </TABLE> <CHAMP> … </CHAMP> <VALUE> … </VALUE> <CONDITION> … </CONDITION> </UPDATE>

3. Le résultat d'un SELECT sera produit en XML :

```
<RESULTAT>
  <TUPLES>
     <TUPLE>
        <CHAMP> Value </CHAMP>
        <CHAMP> Value </CHAMP>
          …
     </TUPLE>
```

```
      <TUPLE>
          <CHAMP> Value </CHAMP>
          <CHAMP> Value </CHAMP>
            ...
       </TUPLE>
        ...
  </TUPLES>
</RESULTAT>
```

4. Les requêtes en XML doivent être signées pour être validées par l'application. Les résultats en XML seront également signés.

L'application java à implémenter doit être organisée en quatre classes :

a) **Main.java** est le programme principal qui doit afficher le menu
   (R)echercher
   (I)nsérer
   (M)ettre à jour
   (E)ffacer
   (Q)uitter
   pour lancer les fonctionnalités de l'application.

b) **Rechercher.java** charge et valide le fichier XML signé, puis le transforme en requête sql qui sera appliquée sur la BD oracle. Le résultat de la recherche sera stocké dans un fichier XML signé.

c) Les classes **Inserer.java**, **Maj.java** et **Effacer.java** chargent et valident le fichier XML signé pour le transformer en sql et l'exécuter sur la BD oracle.

Les détails sur la BD relationnelle à utiliser sont donnés à la page suivante.

Travail à rendre

1. Quatre fichiers code source (**Main.java**, **Rechercher.java**, **Inserer.java**, **Maj.java** et **Effacer.java**) bien commentés.

2. Des exemples de fichiers XML signés pour les quatre fonctionnalités.

Mettre tous les fichiers dans un répertoire portant vos noms, et compresser ce répertoire au format **ZIP** dont le nom de fichier doit suivre le format suivant :

Groupe_NomBinôme1_NomBinôme2.zip

Enfin, envoyer le fichier compressé à votre enseignant.

La coupe du monde de rugby

On suppose que l'on veut modéliser les matches de la coupe du monde de rugby. Pour cela, on considère les données et les contraintes suivantes. L'événement (la coupe du monde) se compose d'un certain nombre de matches. Un match oppose deux équipes, celle qui est sup- posée recevoir, et celle qui est supposée être reçue. Un match se déroule à une date donnée dans un stade. Chaque stade est dans une ville de France, possède un nom, et une capacité donnée, mais chaque match accueille un nombre donné de spectateurs. On désigne les stades et les matches par un numéro qui permet de les identifier de manière unique. Pour chaque match, on veut connaître le nombre de points de l'équipe qui reçoit, ainsi que le nombre d'essais. On souhaite avoir les mêmes informations pour l'équipe qui est accueillie. La coupe du monde oppose des équipes nationales. Chaque équipe possède un code qui per- met de la différencier des autres équipes. Les informations que l'on souhaite conserver sont le pays d'une équipe, la couleur du maillot porté par les joueurs (on ne prend pas en compte les changements de maillots), et l'entraîneur de chaque équipe. Bien sûr, une équipe est composée de plusieurs joueurs qui occupent un poste donné sur le terrain (numéro du poste, et libellé). Les postes sont les mêmes pour toutes les équipes. Un joueur possède un numéro, un nom, et un prénom, et il joue à un poste donné. Au départ d'un match, un joueur est titulaire ou remplaçant, et il joue un certain temps (il peut sortir en cours de partie, ou bien jouer tout le match). Durant le match, un joueur peut marquer un certains nombre de points, que cela soit des essais ou des coups de pied. L'organisation de la coupe du monde gère également les arbitres qui vont arbitrer les différents matches. Un arbitre possède un numéro, et les informations concernant un arbitre sont son nom et son prénom, et sa nationalité (car un arbitre d'une nationalité donnée ne peut pas arbitrer l'équipe de son pays).

Pour réaliser ce TP, voici des liens utiles :

1. http://www.loria.fr/~imine/cours/AppBD/Data.zip
   pour récupérer la structure des tables ainsi que les données à insérer.

2. http://www.loria.fr/~imine/ojdbc6.jar
   pour télécharger le driver pour jdbc.

3. http://www.loria.fr/~imine/cours/JDBC.pdf
   pour télécharger un cours sur jdbc.