Groupe de travail Réseau
Request for Comments : 3275
RFC rendue obsolète : 3075
Catégorie : En cours de normalisation
Traduction Claude Brière de L'Isle

D. Eastlake 3rd, Motorola
J. Reagle, W3C
D. Solo, Citigroup
mars 2002

Syntaxe et traitement des signatures XML (Langage de balisage extensible)

Statut de ce mémoire

Le présent document spécifie un protocole Internet en cours de normalisation pour la communauté de l'Internet, et appelle à des discussions et des suggestions pour son amélioration. Prière de se reporter à l'édition actuelle du STD 1 "Normes des protocoles officiels de l'Internet" pour connaître l'état de normalisation et le statut de ce protocole. La distribution du présent mémoire n'est soumise à aucune restriction.

(La présente traduction incorpore la correction des erreurs mentionnées à la date du 11/07/2012 dans l'errata situé à www.w3.org/2001/10/xmldsig-errata, mais pas les précisions et avertissements qui y figurent.)

Résumé

Le présent document spécifie les règles de traitement et la syntaxe de la signature numérique du langage de balisage extensible (XML, *Extensible Markup Language*). Les signatures XML fournissent les services d'intégrité, d'authentification de message, et/ou d'authentification du signataire pour des données de tout type, qu'elles soient localisées au sein du XML qui comporte la signature ou ailleurs.

Table des matières

1. Introduction

Le présent document spécifie la syntaxe et les règles de traitement XML pour la création et la représentation des signatures numériques. Les signatures XML peuvent s'appliquer à tout contenu numérique (objet de données) y compris XML. Une signature XML peut être appliquée au contenu d'une ou plusieurs ressources. Les signatures enveloppées ou enveloppantes sont parmi les données au sein du même document XML que la signature ; les signatures détachées sont parmi les données externes à l'élément de signature. Plus précisément, la présente spécification définit un type d'élément Signature XML et une application XML de signature ; les exigences de conformité pour chacune sont spécifiées au moyen de définitions sous forme respectivement de schéma et de prose. La présente spécification comporte aussi d'autres types utiles qui identifient les méthodes pour faire référence aux collections de ressources, d'algorithmes, et d'informations de matériel de clé et de gestion.

La signature XML est une méthode qui associe une clé à des données référencées (octets) ; elle ne spécifie pas de façon normative comment les clés sont associées aux personnes ou institutions, ni la signification des données qui sont référencées et signées. Par conséquent, bien que cette spécification soit une composante importante des applications XML sécurisées, elle n'est pas suffisante par elle-même pour régler tous les problèmes de sécurité d'application/confiance, particulièrement par rapport à l'utilisation de la signature XML (ou autres formats de données) comme base d'une communication et d'un accord entre humains. Une telle application doit spécifier des clés supplémentaires, un algorithme, des exigences de traitement et de résultats. Pour des informations complémentaires, prière de se reporter à la section 8 Considérations pour la sécurité.

1.1 Conventions rédactionnelles et de conformité

Pour des raisons de lisibilité, pour faire court, et pour des raisons historiques, le présent document utilise le terme "signature" pour se référer d'une façon générale à des valeurs d'authentification numériques de tous types. Évidemment, le terme est aussi utilisé pour se référer strictement aux valeurs d'authentification qui se fondent sur des clés publiques et qui fournissent l'authentification du signataire. Pour parler spécifiquement des valeurs d'authentification fondées sur des codes de clés secrètes symétriques, on utilisera les termes d'authentifiant ou de codes d'authentification. (Voir au paragraphe 8.3, "Vérifier le modèle de sécurité".)

La présente specification présente un schéma XML [XML-sch] et DTD [XML]. La définition du schéma est normative.

Dans le présent document, les mots clé "DOIT", "NE DOIT PAS", "EXIGÉ", "DEVRAIT" NE DEVRAIT PAS", "RECOMMANDÉ", "NON RECOMMANDÉ", "PEUT", et "FACULTATIF" doivent être interprétés comme décrit dans le BCP 14, [RFC2119] : "ils DOIVENT être utilisés dans les seuls cas où ils sont réellement exigés pour l'inter fonctionnement ou pour limiter un comportement potentiellement dommageable (par exemple, de limiter les retransmissions)".

Par conséquent, on utilise ces mots clés en majuscules pour spécifier sans ambiguité les exigences envers des caractéristiques du protocole et des applications et des comportements qui affectent l'interopérabilité et la sécurité des mises en œuvre. Ces mots clés ne sont pas utilisés (en majuscule) pour décrire la grammaire XML ; les définitions de schéma décrivent sans ambiguité de telles exigences et on souhaite réserver l'importance de ces termes pour les descriptions en langage naturel des protocoles et des caractéristiques. Par exemple, un attribut XML pourrait être décrit comme étant "facultatif." La conformité aux espaces de noms dans la spécification XML [XML-ns] est décrite comme "EXIGÉ."

1.2 Philosophie du document

La philosophie et les exigences de la présente spécification sont traitées dans le document "Exigences pour les signatures XML" [RFC2807].

1.3 Versions, espaces de noms et identifiants

Il n'existe aucune disposition pour un numéro de version explicite dans cette syntaxe. Si une version future est nécessaire, elle utilisera un espace de nom différent. L'URI de l'espace de nom XML [XML-ns] qui DOIT être utilisé par les mises en œuvre de cette spécification (datée) est :

xmlns="http://www.w3.org/2000/09/xmldsig#"

Cet espace de nom est aussi utilisé comme préfixe des identifiants d'algorithme de cette spécification. Alors que les applications DOIVENT prendre en charge XML et les espaces de nom XML, l'utilisation des entités internes [XML] ou de notre préfixe d'espace de nom "dsig" XML et des conventions de valeur par défaut/portée est FACULTATIVE ; on utilise ces facilités pour fournir des exemples compacts et lisibles.

La présente spécification utilise des identifiants de ressources uniformes (URI, *Uniform Resource Identifier*) [RFC2396] pour identifier les ressources, les algorithmes, et la sémantique. L'URI dans la déclaration d'espace de nom ci-dessus est aussi utilisé comme préfixe pour les URI qui se soumettent à la présente spécification. Pour les ressources qui échappent au contrôle de la présente spécification, nous utilisons la désignation de nom de ressource uniforme (URN, *Uniform Resource Name*) [RFC2141] ou localisateur uniforme de ressource (URL, *Uniform Resource Locator*) [RFC1738] définie par sa spécification normative externe. Si une spécification externe n'a pas elle même alloué d'identifiant de ressource uniforme, nous allouons un identifiant dans notre propre espace de noms. Par exemple :

SignatureProperties est identifié et défini par l'espace de noms de la présente spécification :
http://www.w3.org/2000/09/xmldsig#SignatureProperties
XSLT est identifié et défini par un URI externe : http://www.w3.org/TR/1999/REC-xslt-19991116
SHA1 est identifié via l'espace de nom de cette spécification et défini via une référence normative :
http://www.w3.org/2000/09/xmldsig#sha1
(FIPS PUB 180-1. Secure Hash Standard. Ministère américain du Commerce/Institut National des Normes et Technologies.)

Finalement, pour fournir des déclarations concises d'espace de noms on utilise parfois des entités XML internes [XML] au sein des URI. par exemple :

```
<?xml version='1.0'?>
<!DOCTYPE Signature SYSTEM
"xmldsig-core-schema.dtd" [ <!ENTITY dsig
"http://www.w3.org/2000/09/xmldsig#"> ]>
<Signature xmlns="&dsig;" Id="MyFirstSignature">
<SignedInfo>
...
```

1.4 Remerciements

Des remerciement chaleureux sont adressés pour leurs contributions à la présente spécification aux membres suivants du groupe de travail :

* Mark Bartel, Accelio (auteur)
* John Boyer, PureEdge (auteur)
* Mariano P. Consens, University of Waterloo
* John Cowan, Reuters Health
* Donald Eastlake 3rd, Motorola (président, auteur/éditeur)
* Barb Fox, Microsoft (auteur)
* Christian Geuer-Pollmann, University Siegen
* Tom Gindin, IBM
* Phillip Hallam-Baker, VeriSign Inc

* Richard Himes, US Courts
* Merlin Hughes, Baltimore
* Gregor Karlinger, IAIK TU Graz
* Brian LaMacchia, Microsoft (auteur)
* Peter Lipp, IAIK TU Graz
* Joseph Reagle, W3C (président, auteur/éditeur)
* Ed Simon, XMLsec (auteur)
* David Solo, Citigroup (auteur/éditeur)
* Petteri Stenius, DONE Information, Ltd
* Raghavan Srinivas, Sun
* Kent Tamura, IBM
* Winchel Todd Vincent III, GSU
* Carl Wallace, Corsec Security, Inc.
* Greg Whitehead, Signio Inc.

Et pour leurs commentaires sur le projet les personnes suivantes :

* Dan Connolly, W3C
* Paul Biron, Kaiser Permanente, au nom du groupe de travail XML Schema.
* Martin J. Duerst, W3C; et Masahiro Sekiguchi, Fujitsu; au nom du groupe de travail Internationalization.
* Jonathan Marsh, Microsoft, au nom du groupe de travail Extensible Stylesheet Language.

1.5 Statut au W3C

La recommandation du consortium World Wide Web correspondant à la présente RFC est à :

http://www.w3.org/TR/2002/REC-xmldsig-core-20020212/

2. Généralités et exemples sur les signatures

La présente section donne une vue générale et des exemples de syntaxe de signature numérique XML. Le traitement spécifique est donné dans les Règles de traitement (section 3). La syntaxe formelle se trouve dans Cœur de la syntaxe de Signature (section 4) et dans Syntaxe additionnelle de Signature (section 5).

Dans cette section, une représentation informelle et des exemples sont utilisés pour décrire la structure de la syntaxe de signature XML. Cette représentation et ces exemples peuvent omettre des attributs, des détails et des caractéristiques potentielles qui seront expliqués en détail ultérieurement.

Les signatures XML sont appliquées à des contenus numériques (objets de données) arbitraires via un adressage indirect. Les objets de données sont résumés, la valeur résultante est placée dans un élément (avec d'autres informations) et cet élément est alors résumé et signé cryptographiquement. Les signatures numériques XML sont représentées par l'élément Signature qui a la structure suivante (où "?" note zéro ou une occurrence, "+" note une ou plusieurs occurrences, et "*" note zéro, une ou plusieurs occurrences) :

```
<Signature ID?>
<SignedInfo>
<CanonicalizationMethod/>
<SignatureMethod/>
(<Reference URI? >
(<Transforms>)?
<DigestMethod>
<DigestValue>
</Reference>)+
</SignedInfo>
<SignatureValue>
(<KeyInfo>)?
(<Object ID?>)*
</Signature>
```

Les signatures se rapportent aux objets de données via des URI [RFC2396]. Au sein d'un document XML, les signatures se

rapportent aux objets de données locaux via des identitifants de fragment. De telles données locales peuvent être incluses au sein d'une signature enveloppante ou peuvent inclure une signature enveloppée. Les signatures détachées sont sur des ressources externes du réseau ou des objets de données locaux qui résident au sein du même document XML comme éléments frères ; dans ce cas, la signature n'est ni un attribut enveloppant (la signature est le parent) ni un attribut enveloppé la signature est fille). Comme un élément Signature (et son identifiant de valeur/nom) peut coexister ou être combiné avec d'autres éléments (et leurs identifiants) au sein d'un seul document XML, il faut veiller à choisir des noms tels qu'il n'y ait pas de collisions ultérieures qui violent les contraintes de validité sur l'unicité de l'identifiant [XML].

2.1 Exemple simple (Signature, SignedInfo, méthodes et références)

L'exemple suivant est une signature détachée du contenu de HTML4 dans la spécification XML.

```
[s01] <Signature Id="MyFirstSignature" xmlns="http://www.w3.org/2000/09/xmldsig#">
[s02]   <SignedInfo>
[s03]   <CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
[s04]   <SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#dsa-sha1"/>
[s05]   <Reference URI="http://www.w3.org/TR/2000/REC-xhtml1-20000126/">
[s06]     <Transforms>
[s07]       <Transform Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
[s08]     </Transforms>
[s09]     <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
[s10]     <DigestValue>j6lwx3rvEPO0vKtMup4NbeVu8nk=</DigestValue>
[s11]   </Reference>
[s12] </SignedInfo>
[s13]   <SignatureValue>MC0CFFrVLtRlk=...</SignatureValue>
[s14]   <KeyInfo>
[s15a]    <KeyValue>
[s15b]      <DSAKeyValue>
[s15c]        <P>...</P><Q>...</Q><G>...</G><Y>...</Y>
[s15d]      </DSAKeyValue>
[s15e]    </KeyValue>
[s16]   </KeyInfo>
[s17] </Signature>
```

[s02-12] L'élément exigé SignedInfo est l'information qui est en fait signée. Le cœur de la validation de SignedInfo consiste en deux processus obligatoires : la validation de la signature sur SignedInfo et la validation de chaque résumé Reference au sein de SignedInfo. Noter que les algorithmes utilisés pour le calcul de SignatureValue sont aussi inclus dans les informations signées alors que l'élément SignatureValue est en dehors de SignedInfo.

[s03] CanonicalizationMethod est l'algorithme qui est utilisé pour canoniser l'élément SignedInfo avant qu'il soit résumé au titre de l'opération de signature. Noter que cet exemple, et tous les exemples de la présente spécification, ne sont pas en forme canonique.

[s04] SignatureMethod est l'algorithme qui est utilisé pour convertir le SignedInfo canonisé en SignatureValue. C'est une combinaison d'un algorithme de résumé et d'un algorithme dépendant de la clé et éventuellement d'autres algorithmes comme de bourrage, par exemple RSA-SHA1. Les noms d'algorithmes sont signés pour résister aux attaques fondées sur la substitution d'un algorithme plus faible. Pour promouvoir l'interopérabilité des applications, on spécifie un ensemble d'algorithmes de signatures qui DOIVENT être mis en œuvre, bien que leur utilisation soit à la discrétion du créateur de la signature. On spécifie des algorithmes supplémentaires comme RECOMMANDÉS or FACULTATIFS à la mise en œuvre ; la conception permet aussi des algorithmes arbitraires spécifiés par l'utilisateur.

[s05-11] Chaque élément Reference comporte la méthode de résumé et la valeur de résumé résultante calculée sur l'objet de données identifié. Il peut aussi inclure des transformations qui produisent l'entrée à l'opération de résumé. Un objet de données est signé en calculant sa valeur de résumé et une signature sur cette valeur. La signature est ultérieurement vérifiée par la validation de la référence et de la signature.

[s14-16] KeyInfo indique la clé à utiliser pour valider la signature. Les formes possibles pour l'identification incluent des algorithmes et informations de certificats, de noms de clé, et d'accord de clés — on n'en définit que quelques uns. KeyInfo est facultatif pour deux raisons. D'abord, le signataire peut ne pas souhaiter révéler les informations de clés à toutes les parties qui traitent le document. Ensuite, les informations peuvent être connues au sein du

contexte de l'application et n'ont pas besoin d'être représentées explicitement. Comme KeyInfo est en dehors de SignedInfo, si le signataire souhaite lier les informations de clé à la signature, une référence peut facilement identifier et inclure le KeyInfo au titre de la signature.

2.1.1 Précisions sur la référence

```
[s05] <Reference URI="http://www.w3.org/TR/2000/REC-xhtml1-20000126/">
[s06]  <Transforms>
[s07]   <Transform Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
[s08]  </Transforms>
[s09]  <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
[s10]  <DigestValue>j6lwx3rvEPO0vKtMup4NbeVu8nk=</DigestValue>
[s11] </Reference>
```

[s05] L'attribut facultatif d'URI de Reference identifie l'objet de données à signer. Cet attribut peut être omis sur au plus une Reference dans une Signature. (Cette limitation est imposée afin d'assurer que références et objets peuvent se correspondre sans ambiguïté.)

[s05-08] Cette identification, ainsi que les transformations, est une description fournie par le signataire de la façon dont on a obtenu l'objet de données signé dans la forme sous laquelle il a été résumé (c'est-à-dire, le contenu résumé). Le vérificateur peut obtenir le contenu résumé par une autre méthode pour autant que le résumé corresponde. En particulier, le vérificateur peut obtenir le contenu à partir d'une localisation différente, telle qu'une mémoire locale, par opposition à celle spécifiée dans l'URI.

[s06-08] Transforms est une liste ordonnée facultative d'étapes de traitements qui ont été appliquées au contenu de la ressource avant qu'il soit résumé. Transforms peut inclure des opérations telles que la canonisation, le codage/décodage (y compris la compression/inflation), XSLT, XPath, la validation de schéma XML, ou XInclude. Les transformations XPath permettent au signataire de déduire un document XML qui omet des portions du document source. Par conséquent, ces portions exclues peuvent changer sans affecter la validité de la signature. Par exemple, si la ressource signée inclut la signature elle-même, une telle transformation doit être utilisée pour exclure la valeur de la signature de son propre calcul. Si aucun élément Transforms n'est présent, le contenu de la ressource est résumé directement. Alors que le groupe de travail a spécifié des algorithmes obligatoires (et facultatifs) de canonisation et de décodage, les transformations spécifiées par l'utilisateur sont permises.

[s09-10] DigestMethod est l'algorithme appliqué aux données après l'application de Transforms (si elle est spécifiée) pour donner DigestValue. La signature de DigestValue est ce qui lie le contenu d'une ressource à la clé du signataire.

2.2 Exemple étendu (propriétés d'objet et de signature)

La présente spécification ne traite pas des mécanismes pour faire des déclarations ou assertions. Le présent document définit plutôt ce que signifie pour quelque chose d'être signé par une signature XML (intégrité, authentification du message, et/ou authentification du signataire). Les applications qui souhaitent représenter une autre sémantique doivent s'appuyer sur d'autres technologies, telles que [XML], [RDF]. Par exemple, une application pourrait utiliser un attribut foo:assuredby au sein de son propre marquage pour référencer un élément Signature. Par conséquent, c'est l'application qui doit comprendre et savoir comment prendre des décisions de confiance en fonction de la validité de la signature et de la signification de la syntaxe assuredby. On définit aussi un type d'élément SignatureProperties à inclure dans les assertions sur la signature elle-même (par exemple, sémantique de signature, heure de la signature ou numéro de série du matériel utilisé dans le processus de chiffrement). De telles assertions peuvent être signées en incluant une référence pour les SignatureProperties dans SignedInfo. Alors que l'application signataire devrait être très prudente au sujet de ce qu'elle signe (elle devrait comprendre ce qui est dans la SignatureProperty) une application receveur n'a pas l'obligation de comprendre cette sémantique (bien que son moteur de confiance parent puisse le souhaiter). Tout contenu relatif à la génération de la signature peut être situé au sein de l'élément SignatureProperty. L'attribut obligatoire Target *(cible)* fait références à l'élément Signature auquel la propriété s'applique.

Considérons l'exemple précédant avec une référence supplémentaire à un objet local qui comporte un élément SignatureProperty. (Une telle signature ne serait pas seulement détachée [p02] mais aussi enveloppante [p03].)

```
[ ] <Signature Id="MySecondSignature" ...>
```

```
[p01]  <SignedInfo>
[  ]   ...
[p02]   <Reference URI="http://www.w3.org/TR/xml-stylesheet/">
[  ]   ...
[p03]   <Reference URI="#AMadeUpTimeStamp"
[p04]   Type="http://www.w3.org/2000/09/xmldsig#SignatureProperties">
[p05]    <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
[p06]    <DigestValue>k3453rvEPO0vKtMup4NbeVu8nk=</DigestValue>
[p07]   </Reference>
[p08]  </SignedInfo>
[p09]  ...
[p10]  <Object>
[p11]   <SignatureProperties>
[p12]    <SignatureProperty Id="AMadeUpTimeStamp" Target="#MySecondSignature">
[p13]      <timestamp xmlns="http://www.ietf.org/rfcXXXX.txt">
[p14]        <date>19990908</date>
[p15]        <time>14:34:34:34</time>
[p16]      </timestamp>
[p17]    </SignatureProperty>
[p18]   </SignatureProperties>
[p19]  </Object>
[p20]</Signature>
```

[p04] L'attribut Type facultatif de Reference donne des informations sur la ressource identifiée par l'URI. En particulier, il peut indiquer qu'il est un élément Object, SignatureProperty, ou Manifest. Cela peut être utilisé par les applications pour initier un traitement spécial de certains éléments Reference. Les références à un élément de données XML au sein d'un élément Object DEVRAIENT identifier l'élément réel qui est pointé. Lorsque le contenu de l'élément n'est pas XML (il est peut-être en données binaires ou codées) la référence devrait identifier l'objet, et le type de Reference, si il est donné, DEVRAIT indiquer Object. Noter que Type est pour information et aucune action fondée sur lui ni vérification de son adéquation n'est exigée par le comportement type.

[p10] Object est un élément facultatif pour inclure des objets de données au sein de l'élément de signature ou ailleurs. Object peut facultativement être muni d'un type et/ou codé.

[p11-18] Les propriétés de la signature, comme l'heure de signature, peuvent facultativement être signées en les identifiant au sein d'une Reference. (Ces propriétés sont traditionellement appelés "attributs" de signature bien que ce terme n'ait pas de relation avec le terme XML "attribute".)

2.3 Exemple étendu (Objet et manifeste)

L'élément Manifest est fourni pour satifaires à des exigences supplémentaires non directement traitées par les parties obligatoires de la présente spécification. Voici deux exigences et la façon dont Manifest les satisfait.

D'abord, les applications ont fréquemment besoin de signer efficacemment plusieurs objets de données même lorsque l'opération de signature elle-même est une coûteuse signature à clé publique. Cette exigence peut être satisfaite en incluant plusieurs éléments Reference au sein de SignedInfo car l'inclusion de chaque résumé sécurise les données résumées. Cependant, certaines applications peuvent ne pas vouloir le comportement de validation usuel associé à cette approche parce que il exige que chaque Reference au sein de SignedInfo subisse la validation de référence – les éléments DigestValue sont vérifiés. Ces applications peuvent souhaiter se réserver pour elles seules la logique de la décision de validation de référence. Par exemple, une application pourrait recevoir un élément de signature SignedInfo valide qui comporte trois éléments Reference. Si un seul Reference échoue (l'objet de données identifié lors du résumé ne donne pas la DigestValue spécifiée) la signature va échouer au cœur de la validation. Cependant, l'application peut souhaiter traiter la signature sur les deux éléments Reference valides comme valide ou prendre des actions différentes selon celle qui échoue. Pour réaliser cela, SignedInfo ferait référence à un élément Manifest qui contient un ou plusieurs éléments Reference (avec la même structure que celle dans SignedInfo). Ensuite, la validation de référence de Manifest est sous le contrôle de l'application.

Ensuite, considérons une application où de nombreuses signatures (utilisant des clés différentes) sont appliquées à un grand nombre de documents. Une solution inefficace est d'avoir une signature distincte (par clé) appliquée de façon répétitive à un gros élément SignedInfo (avec de nombreuses références) ; c'est du gachis et c'est redondant. Une solution plus efficace est d'inclure plusieurs références dans un seul Manifest qui est alors référencé à partir de plusieurs éléments Signature.

L'exemple ci-dessous comporte une Reference qui signe un Manifest trouvé au sein de l'élément Object.

```
[ ] ...
[m01] <Reference URI="#MyFirstManifest"
[m02]   Type="http://www.w3.org/2000/09/xmldsig#Manifest">
[m03]   <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
[m04]   <DigestValue>345x3rvEPO0vKtMup4NbeVu8nk=</DigestValue>
[m05] </Reference>
[ ] ...
[m06] <Object>
[m07] <Manifest Id="MyFirstManifest">
[m08]   <Reference>
[m09]   ...
[m10]   </Reference>
[m11]   <Reference>
[m12]   ...
[m13]   </Reference>
[m14] </Manifest>
[m15] </Object>
```

3. Règles de traitement

Les paragraphes qui suivent décrivent les opérations à effectuer au titre de la génération et la validation de la signature.

3.1 Génération du cœur

Les étapes EXIGÉES incluent la génération des éléments Reference et la SignatureValue sur SignedInfo.

3.1.1 Génération de référence

Pour chaque objet de données signé :

1. Appliquer les transformations, comme déterminé par l'application, à l'objet de données.
2. Calculer la valeur du résumé sur l'objet de données résultant.
3. Créer un élément Reference, incluant l'identification (facultative) de l'objet de données, tous éléments (facultatif) de transformation, l'algorithme de résumé et la valeur de résumé. (Noter que c'est la forme canonique de ces références qui est signée en 3.1.2 et validée en 3.2.1.)

3.1.2 Génération de signature

1. Créer un élément SignedInfo avec la méthode de signature, la méthode de canonisation et la ou les références.
2. Canoniser puis calculer la valeur de signature sur SignedInfo selon les algorithmes spécifiés dans SignedInfo.
3. Construire les éléments de signature qui incluent SignedInfo, le ou les objets (si désiré, le codage peut être différent de celui utilisé pour signer) les informations de clés (si exigé) et la valeur de signature.

Note : Si Signature comporte des références same-document, la validation [XML] ou [XML-sch] du document peut introduire des changements qui cassent la signature. Par conséquent, les applications devraient faire attention à traiter le document de façon cohérente ou à s'abstenir d'utiliser des contributions externes (par exemple, par défaut et des entités externes).

3.2 Validation du cœur

Les étapes EXIGÉES de la validation du cœur incluent (1) la validation de la référence, la vérification du résumé contenu dans chaque Reference dans SignedInfo, et (2) la validation de signature cryptographique de la signature calculée sur SignedInfo.

Note : Il peut y avoir des signatures valides que certaines applications de signature ne sont pas capables de valider. Les raisons de cela incluent l'échec de la mise en œuvre de parties facultatives de la présente spécification, l'incapacité

ou le refus d'exécuter les algorithmes spécifiés, ou l'incapacité ou le refus de déréférencer les URI spécifiés (certains schémas d'URI peuvent causer des effets colatéraux indésirables), etc.

La comparaison de valeurs dans la validation de références et signatures se fait sur la séquence d'octets numérique (par exemple, d'entiers) ou décodée de la valeur. Des mises en œuvre différentes peuvent produire des valeurs différentes de résumé codé et de signature lors du traitement des mêmes ressources à cause de variations de leur codage, comme une espace blanche accidentelle. Mais, si on utilise la comparaison numérique ou d'octet (il faut en choisir une) sur les deux valeurs déclarée et calculée, ces problèmes sont éliminés.

3.2.1 Validation de référence

1. Canoniser l'élément SignedInfo sur la base de la méthode de canonisation dans SignedInfo.
2. Pour chaque Reference dans SignedInfo :
   2.1 Obtenir l'objet de données à résumer. (par exemple, l'application de signature peut déréférencer l'URI et exécuter les transformations fournies par le signataire dans l'élément Reference, ou elle peut obtenir le contenu par d'autres moyens tels qu'une antémémoire locale.)
   2.2 Résumer l'objet de données résultant en utilisant la méthode de résumé spécifiée dans sa spécification de référence.
   2.3 Comparer la valeur de résumé générée à la valeur de résumé dans la référence SignedInfo ; si il y a la moindre discordance, la validation échoue.

Note : SignedInfo est canonisé dans l'étape 1. L'application doit s'assurer que la méthode de canonisation n'a pas d'effets colatéraux dangereux, tels que la réécriture des URI, (voir au paragraphe 4.3, Méthode de canonisation) et qu'elle voit ce qui est signé, ce qui est la forme canonique.

3.2.2 Validation de Signature

1. Obtenir les informations de clés de KeyInfo ou d'une source externe.
2. Obtenir la forme canonique de la méthode de signature en utilisant la méthode de canonisation et utiliser le résultat (et les KeyInfo obtenues précédemment) pour confirmer la valeur de signature sur l'élément SignedInfo.

Note : KeyInfo (ou une version transformée de KeyInfo) peut être signé via un élément Reference. La transformation et la validation de cette référence (3.2.1) est orthogonale à la validation de signature qui utilise les KeyInfo comme analysées.

De plus, l'URI SignatureMethod peut avoir été altéré par la canonisation de SignedInfo (par exemple, la mise en absolus d'URI relatifs) et c'est la forme canonique qui DOIT être utilisée. Cependant, la canonisation exigée de la [RFC3076] de la présente spécification ne change pas les URI.

4. Cœur de la syntaxe de signature

La structure générale d'une signature XML est décrite à la section 2. La présente section donne la syntaxe détaillée des caractéristiques centrales de la signature. Les caractéristiques décrites dans la présente section sont de mise en œuvre obligatoire, sauf indication contraire. La syntaxe est définie via des DTD (Data Type Declaration, *déclaration de type de données*) et par [XML-sch] avec le préambule, déclaration, et entité interne XML suivants.

Définition du schéma :

```
    <?xml version="1.0" encoding="utf-8"?>
    <!DOCTYPE schema
      PUBLIC "-//W3C//DTD XMLSchema 200102//EN"
  "http://www.w3.org/2001/XMLSchema.dtd"
     [
      <!ATTLIST schema
        xmlns:ds CDATA #FIXED "http://www.w3.org/2000/09/xmldsig#">
      <!ENTITY dsig 'http://www.w3.org/2000/09/xmldsig#'>
      <!ENTITY % p ">
      <!ENTITY % s ">
     ]>

    <schema xmlns="http://www.w3.org/2001/XMLSchema"
```

```
        xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
        targetNamespace="http://www.w3.org/2000/09/xmldsig#"
        version="0.1" elementFormDefault="qualified">
```

DTD:

```
    <!--

Les déclarations d'entité suivantes permettent un contenu externe/flexible dans le modèle de contenu Signature.

#PCDATA émule le schéma : chaîne ; lorsque combiné avec les types element, il émule le schéma mixte="vrai".

%foo.ANY permet à l'utilisateur d'inclure ses propres types element à partir d'autres espaces de noms, par exemple :
        <!ENTITY % KeyValue.ANY '| ecds:ECDSAKeyValue'>
        ...
        <!ELEMENT ecds:ECDSAKeyValue (#PCDATA)  >

    -->

    <!ENTITY % Object.ANY ">
    <!ENTITY % Method.ANY ">
    <!ENTITY % Transform.ANY ">
    <!ENTITY % SignatureProperty.ANY ">
    <!ENTITY % KeyInfo.ANY ">
    <!ENTITY % KeyValue.ANY ">
    <!ENTITY % PGPData.ANY ">
    <!ENTITY % X509Data.ANY ">
    <!ENTITY % SPKIData.ANY ">
```

4.0.1 Type simple ds:CryptoBinary

La présente spécification définit le type simple ds:CryptoBinary pour représenter des entiers de longueur arbitraire (par exemple, "bignums") en XML comme des chaînes d'octets. La valeur d'entier est d'abord convertie en une chaîne binaire "gros boutienne". La chaîne binaire est ensuite bourrée avec des bits à zéro en tête afin que le nombre total de bits == 0 mod 8 (afin qu'il y ait un nombre entier d'octets). Si la chaîne binaire contient des octets entiers en tête qui sont à zéro, ils sont retirés (afin que l'octet de poids fort soit toujours différent de zéro). Cette chaîne d'octets est alors codée en base64 [RFC2045]. (La conversion d'entier en chaîne d'octets est équivalente au I2OSP [1363] de IEEE 1363 avec une longueur minimale).

Ce type est utilisé par les valeurs "bignum" telles que RSAKeyValue et DSAKeyValue. Si une valeur peut être du type base64Binary ou ds:CryptoBinary, elle est définie comme base64Binary. Par exemple, si l'algorithme de signature est RSA ou DSA, SignatureValue représente alors un bignum et pourrait être ds:CryptoBinary. Cependant, si HMAC-SHA1 est l'algorithme de signature, SignatureValue pourrait alors avoir des octets à zéro en tête qui doivent être préservés. Donc, SignatureValue est défini génériquement comme du type base64Binary.

Définition du schéma :

```
    <nom de simpleType ="CryptoBinary">
      <restriction base="base64Binary">
      </restriction>
    </simpleType>
```

4.1 Élément Signature

L'élément Signature est l'élément racine d'une signature XML. La mise en œuvre DOIT générer des éléments Signature valides de schéma souple [XML-sch] comme spécifié par le schéma suivant :

Définition du schéma :

```
    <nom d'élément = type "Signature" ="ds:SignatureType"/>
    <nom de complexType ="SignatureType">
      <sequence>
        <element ref="ds:SignedInfo"/>
        <element ref="ds:SignatureValue"/>
```

```
      <element ref="ds:KeyInfo" minOccurs="0"/>
      <element ref="ds:Object" minOccurs="0" maxOccurs=«non limité»/>
    </sequence>
    <nom d'attribut = type "Id" = "ID" usage="optionnel"/>
  </complexType>
```

DTD :

```
  <!ELEMENT Signature (SignedInfo, SignatureValue, KeyInfo?,
 Object*)  >
  <!ATTLIST Signature
   xmlns   CDATA   #FIXED 'http://www.w3.org/2000/09/xmldsig#'
   Id      ID  #IMPLICITE >
```

4.2 Élément SignatureValue

L'élément SignatureValue contient la valeur réelle de la signature numérique ; il est toujours codé en utilisant base64 [RFC2045]. Bien qu'on identifie deux algorithmes SignatureMethod, l'un de mise en œuvre obligatoire et l'autre de mise en œuvre facultative, des algorithmes spécifiés par l'utilisateur peuvent aussi être utilisés.

Définition du schéma :

```
  <nom d'élément="SignatureValue" type="ds:SignatureValueType"/>
  <complexType name="SignatureValueType">
    <simpleContent>
      <extension base="base64Binary">
        <nom d'attribut="Id" type="ID" usage="optionnel"/>
      </extension>
    </simpleContent>
  </complexType>
```

DTD :

```
  <!ELEMENT SignatureValue (#PCDATA) >
  <!ATTLIST SignatureValue
          Id  ID      #IMPLICITE>
```

4.3 Élément SignedInfo

La structure de SignedInfo inclut l'algorithme de canonisation, un algorithme de signature, et une ou plusieurs références. L'élément SignedInfo peut contenir un attribut ID facultatif qui va permettre d'être référencé par d'autres signatures et objets.

SignedInfo n'inclut pas de signature explicite ou de propriétés de résumé (comme l'heure de calcul, le numéro de série de l'appareil cryptographique, etc.). Si une application a besoin d'associer des propriétes à la signature ou résumé, elle peut inclure de telles informations dans un élément SignatureProperties au sein d'un élément Object.

Définition du schéma :

```
  <nom d'élément="SignedInfo" type="ds:SignedInfoType"/>      <nom de complexType ="SignedInfoType">
    <sequence>
      <element ref="ds:CanonicalizationMethod"/>
      <element ref="ds:SignatureMethod"/>
      <element ref="ds:Reference" maxOccurs=«non limité»/>
    </sequence>
    <nom d'attribut="Id" type="ID" usage="optionnel"/>
  </complexType>
```

DTD :

```
  <!ELEMENT SignedInfo (CanonicalizationMethod,
   SignatureMethod,  Reference+)  >
  <!ATTLIST SignedInfo
   Id   ID      #IMPLICITE
```

4.3.1 Élément CanonicalizationMethod

CanonicalizationMethod est un élément exigé qui spécifie l'algorithme de canonisation appliqué à l'élément SignedInfo avant d'effectuer les calculs de signature. Cet élément utilise la structure générale des algorithmes décrite dans "Identifiants d'algorithme et exigences de mise en œuvre" (paragraphe 6.1). Les mises en œuvre DOIVENT prendre en charge les algorithmes de canonisation EXIGÉS.

Des solutions de remplacement aux algorithmes de canonisation EXIGÉS (paragraphe 6.5) tels que XML canonique avec commentaires (paragraphe 6.5.1) ou une canonisation minimale (telle que CRLF et normalisation du jeu de caractères) peuvent être explicitement spécifiées mais NE SONT PAS EXIGÉES. Par conséquent, leur utilisation peut ne pas inter opérer avec d'autres applications qui ne prennent pas en charge l'algorithme spécifié (voir la section 7 "Canonisation XML et contraintes de syntaxe"). Les questions de sécurité peuvent aussi survenir dans le traitement des processus et commentaires d'entité si des algorithmes de canonisation sans capacité XML ne sont pas encadrés de la façon appropriée (voir au paragraphe 8.2 : Seul ce qui est "vu" devrait être signé).

La façon dont l'élément SignedInfo est présenté à la méthode de canonisation dépend de cette méthode. Ce qui suit s'applique aux algorithmes qui traitent XML comme des nœuds ou des caractères :

* Les mises en œuvre de canonisation fondées sur XML DOIVENT être approvisionnées avec un ensemble-nœud [XPath] formé à l'origine à partir du document qui contient les SignedInfo et indiquant réellement les SignedInfo, ses descendants, et les nœuds d'attribut et d'espace de noms de SignedInfo et de ses éléments descendants.

* Les algorithmes de canonisation fondés sur le texte (comme CRLF et normalisation de jeu de caractères) devraient disposer des octets UTF-8 qui représentent l'élément SignedInfo bien formé, à partir du premier caractère jusqu'au dernier caractère de la représentation XML, inclus. Cela inclut le texte entier des étiquettes de début et de fin de l'élément SignedInfo ainsi que toutes les balises et données de caractères descendants (c'est-à-dire, le texte) entre ces étiquettes. L'utilisation de la canonisation fondée sur le texte de SignedInfo N'EST PAS RECOMMANDÉE.

On recommande des applications qui mettent en œuvre la canonisation fondée sur le texte plutôt que celles fondées sur XML – telles que les applications à contrainte de ressource – qui génèrent du XML canonisé comme résultat en série afin d'atténuer les problèmes d'interopérabilité et de sécurité. Par exemple, une telle mise en œuvre DEVRAIT (au moins) générer des instances XML autonomes [XML].

Note : L'application de signature doit faire très attention pour accepter et exécuter une méthode de canonisation arbitraire. Par exemple, la méthode de canonisation pourrait réécrire les URI des références en cours de validation. Ou la méthode pourrait transformer massivement les SignedInfo de telle sorte que la validation réussisse toujours (c'est-à-dire, en les convertissant en une signature triviale avec une clé connue sur des données triviales). Comme la méthode de canonisation est à l'intérieur des SignedInfo, dans la forme canonique résultante elle pourrait s'écraser elle-même dans les SignedInfo ou modifier l'élément SignedInfo de telle sorte qu'il apparaisse qu'une fonction de canonisation différente a été utilisée ! Donc, une signature qui paraît authentifier les données désirées avec la clé, méthode de résumé, et méthode de signature désirées peut n'avoir aucun sens si on utilise une méthode de canonisation capricieuse.

Définition du schéma :

```
<nom d'élément="CanonicalizationMethod"
      type="ds:CanonicalizationMethodType"/>
<nom de complexType ="CanonicalizationMethodType" mixte="vrai">
 <sequence>
  <tout espace de noms="##any" minOccurs="0" maxOccurs=«non limité»/>
  <!-- (0,nonlimité) éléments de (1,1) namespace -->
 </sequence>
 <nom d'attribut="Algorithm" type="anyURI" usage="exigé"/>
</complexType>
```

DTD :

```
<!ELEMENT CanonicalizationMethod (#PCDATA %Method.ANY;)* >
<!ATTLIST CanonicalizationMethod
 Algorithm CDATA #EXIGÉ >
```

4.3.2 Élément SignatureMethod

SignatureMethod est un élément exigé qui spécifie l'algorithme utilisé pour la génération et la validation de la signature. Cet algorithme identifie toutes les fonctions cryptographiques impliquées dans l'opération de signature (par exemple, le

hachage, les algorithmes de clé publique, les MAC, le bourrage, etc.). Cet élément utilise la structure générale donnée ici pour les algorithmes décrits au paragraphe 6.1 "Identifiants d'algorithme et exigences de mise en œuvre". Bien qu'il y ait un seul identifiant, cet identifiant peut spécifier un format qui contient plusieurs valeurs de signature distinctes.

Définition du schéma :

```
<nom d'élément="SignatureMethod" type="ds:SignatureMethodType"/>
<nom de complexType ="SignatureMethodType" mixte="vrai">
 <sequence>
  <nom d'élément="HMACOutputLength" minOccurs="0"
       type="ds:HMACOutputLengthType"/>
  <tout espace de noms="##autre" minOccurs="0" maxOccurs=«non limité»/>
  <!-- (0,nonlimité) éléments de (1,1) espace de noms externe -->
 </sequence>
<nom d'attribut="Algorithm" type="anyURI" usage="exigé"/>
</complexType>
```

DTD :

```
<!ELEMENT SignatureMethod
      (#PCDATA|HMACOutputLength %Method.ANY;)* >
<!ATTLIST SignatureMethod
 Algorithm CDATA #EXIGÉ >
```

4.3.3 Élément Reference

Reference est un élément qui peut survenir une ou plusieurs fois. Il spécifie un algorithme de résumé et une valeur de résumé, et facultativement un identifiant de l'objet à signer, le type de l'objet, et/ou une liste de transformations à appliquer avant de faire le résumé. L'identification (de l'URI) et les transformations décrivent comment le contenu résumé (c'est-à-dire, l'entrée dans la méthode de résumé) a été créé. L'attribut Type facilite le traitement des données référencées. Par exemple, bien que la présente spécification ne pose aucune exigence au sujet des données externes, une application peut souhaiter signaler que le référent est un Manifest. Un attribut d'identifiant facultatif permet à une Reference d'être référencée d'ailleurs.

Définition du schéma :

```
<nom d'élément="Reference" type="ds:ReferenceType"/>
<nom de complexType ="ReferenceType">
 <sequence>
  <element ref="ds:Transforms" minOccurs="0"/>
  <element ref="ds:DigestMethod"/>
  <element ref="ds:DigestValue"/>
 </sequence>
 <nom d'attribut="Id" type="ID" usage="optionnel"/>
 <nom d'attribut="URI" type="anyURI" usage="optionnel"/>
 <nom d'attribut="Type" type="anyURI" usage="optionnel"/>
</complexType>
```

DTD :

```
<!ELEMENT Reference (Transforms?, DigestMethod, DigestValue)  >
<!ATTLIST Reference
 Id  ID  #IMPLICITE
 URI CDATA   #IMPLICITE
 Type    CDATA   #IMPLICITE>
```

4.3.3.1 Attribut URI

L'attribut URI identifie un objet de données qui utilise une référence d'URI, comme spécifié dans la [RFC2396]. Le jeu de caractères admis pour les attributs d'URI est le même que pour XML, à savoir [Unicode]. Cependant, certains caractères Unicode sont interdits dans les références d'URI y compris tous les caractères non ASCII et les caractères exclus énumérés au paragraphe 2.4 de la [RFC2396]. Cependant, les signes numéro (#), pourcent (%), et les crochets rectangulaires admis à nouveau dans la [RFC2732] sont permis. L'échappement des caractères interdits doit se faire comme suit :

1. Chaque caractère interdit est converti selon la [RFC3629] en un ou plusieurs octets.
2. Tout octet correspondant à un caractère interdit est esquivé avec le mécanisme d'échappement d'URI (c'est-à-dire, converti en %HH, où HH est la notation hexadécimale de la valeur de l'octet).
3. Le caractère original est remplacé par la séquence de caractères résultante.

Les applications de signature XML DOIVENT être capables d'analyser la syntaxe d'URI. On RECOMMANDE qu'elles soient capables de déréférencer les URI dans le schéma HTTP. Déréférencer un URI dans le schéma HTTP DOIT se conformer aux définitions de code d'état de la [RFC2616] (par exemple, les redirections 302, 305 et 307 sont suivies pour obtenir le corps d'entité d'une réponse 200 de code d'état). Les applications devraient aussi connaître le fait que les paramètres de protocole et les informations d'état (telles que les mouchards HTTP, les profils d'appareil HTML ou la négociation de contenu) peuvent affecter le contenu donné par le déréférencement d'un URI.

Si une ressource est identifiée par plus d'un URI, le plus spécifique devrait être utilisé (par exemple, http://www.w3.org/2000/06/interop-pressrelease.html.en au lieu de http://www.w3.org/2000/06/interop-pressrelease). (Voir au paragraphe 3.2.1 "Validation de référence" des informations complémentaires sur le traitement des références.)

Si l'attribut URI est aussi omis, l'application receveuse est supposée connaître l'identité de l'objet. Par exemple, un protocole léger de données peut omettre cet attribut sachant que l'identité de l'objet fait partie du contexte de l'application. Cet attribut peut être omis d'au plus une référence dans tout SignedInfo, ou Manifest particulier.

L'attribut Type facultatif contient des informations sur le type d'objet à signer. Cela est représenté comme un URI. Par exemple :

Type="http://www.w3.org/2000/09/xmldsig#Object"
Type="http://www.w3.org/2000/09/xmldsig#Manifest"

L'attribut Type s'applique à l'élément sur lequel on pointe, pas à son contenu. Par exemple, une référence qui identifie un élément Objet contenant un élément SignatureProperties est encore du type #Object. Le type d'attribut est indicatif. Aucune validation des informations de type n'est exigée par la présente spécification.

4.3.3.2 Modèle de traitement de Reference

Note : XPath est RECOMMANDÉ. Les applications de signature n'ont pas besoin d'être conformes à la spécification [XPath] pour se conformer à la présente spécification. Cependant, le modèle de données XPath, ses définitions (par exemple, ensemble-nœud) et sa syntaxe sont utilisées dans le présent document afin de décrire ses fonctionnalités pour ceux qui veulent traiter XML comme du XML (au lieu d'octets) au titre de la génération de signature. Pour ceux qui veulent utiliser ces dispositions, une mise en œuvre [XPath] conforme est une façon de les mettre en œuvre, mais elle n'est pas exigée. De telles applications pourraient utiliser une solution de remplacement suffisamment fonctionnelle pour un ensemble-nœud et ne mettre en œuvre que les comportements d'expression XPath EXIGÉS par la présente spécification. Cependant, pour simplifier, on utilise généralement la terminologie XPath sans inclure cette qualification à chaque instant. Les exigences sur les "ensembles-nœuds XPath" peuvent inclure un ensemble-nœud fonctionnel équivalent. Les exigences sur le traitement XPath peuvent inclure des comportements d'application qui sont équivalents au comportement XPath correspondant.

Le type de données du résultat du déréférencement d'URI ou des transformations qui le suivent est un flux d'octets ou un ensemble-nœud XPath.

Les transformations spécifiées dans le présent document sont définies par rapport à l'entrée qu'elles exigent. Le comportement d'application de signature par défaut est le suivant :.

* si l'objet de données est un flux d'octets et si la prochaine transformation exige un ensemble-nœud, l'application de signature DOIT tenter d'analyser les octets donnant l'ensemble-nœud exigé via un traitement [XML] bien formé ;
* si l'objet de données est un ensemble-nœud et si la prochaine transformation exige des octets, l'application de signature DOIT tenter de convertir l'ensemble-nœud en un flux d'octets en utilisant XML canonique [RFC3076].

Les utilisateurs peuvent spécifier d'autres transformations qui outrepassent ces comportements par défaut en transitions entre des transformations qui attendent différentes entrées. Le flux d'octets final contient les octets de données qui vont être sécurisés. L'algorithme de résumé spécifié par DigestMethod est alors appliqué à ces octets de données, résultant en la DigestValue.

Sauf si la référence d'URI est une référence "same-document" comme défini au paragraphe 4.2 de la [RFC2396], le résultat du déréférencement de la référence d'URI DOIT être un flux d'octets. En particulier, un document XML identifié par un URI n'est pas analysé par l'application de signature sauf si l'URI est une référence same-document ou si une transformation qui exige l'analyse XML est appliquée. (Voir au paragraphe 4.3.3.1).)

Lorsque un fragment est précédé d'un URI absolu ou relatif dans la référence d'URI, la signification du fragment est définie par le type MIME de la ressource. Même pour les documents XML, le déréférencement d'URI (y compris le traitement du fragment) peut être fait par un mandataire pour l'application de signature. Donc, la validation de référence peut échouer si le traitement de fragment n'est pas effectué de la façon standard (comme défini au paragraphe suivant pour

les références same-document). Par conséquent, on RECOMMANDE que l'attribut URI n'inclue pas les identifiants de fragment, et qu'un tel traitement soit spécifié comme une transformation XPath supplémentaire.

Lorsque un fragment n'est pas précédé par un URI dans la référence d'URI, les application de signatures XML DOIVENT prendre en charge l'URI nul et le nom Xpointer nu. On RECOMMANDE la prise en charge des XPointers '#xpointer(/)' et '#xpointer(id('ID'))' pour le même document si l'application a aussi l'intention de prendre en charge une canonisation qui préserve les commentaires. (Autrement, URI="#foo" va automatiquement retirer les commentaires avant que la canonisation puisse même être invoquée.) Toute autre prise en charge des XPointers est FACULTATIVE, en particulier, toute prise en charge de nom nu et autres XPointers sur des ressources externes car l'application peut ne pas avoir de contrôle sur la façon dont le fragment est généré (ce qui conduit à des problèmes d'interopérabilité et des échecs de validation).

Les exemples suivants montrent ce que l'attribut URI identifie et comment il est déréférencé :

URI="http://exemple.com/bar.xml"
: Identifie les octets qui représentent la ressource externe 'http://exemple.com/bar.xml', qui est probablement un document XML étant donné son extension de fichier.

URI="http://exemple.com/bar.xml#chapitre1"
: Identifie l'élement avec la valeur d'attribut d'identifiant 'chapitre1' de la ressource XML externe 'http://exemple.com/bar.xml', fournie comme un flux d'octets. Là encore, pour les besoins de l'interopérabilité, l'élément identifié comme 'chapitre1' devrait être obtenu en utilisant une transformation XPath plutôt qu'un fragment d'URI (la résolution du XPointer de nom nu dans des ressources externes n'est pas EXIGÉE dans cette spécification).

URI=""
: Identifie l'ensemble-nœud (moins tout nœud de commentaire) de la ressource XML qui contient la signature.

URI="#chapitre1"
: Identifie un ensemble-nœud contenant l'élément avec la valeur d'attribut d'identifiant 'chapitre1' de la ressource XML qui contient la signature. La signature XML (et ses applications) modifie cet ensemble-nœud pour inclure l'élément plus tous les descendants y compris les espaces de noms et attributs – mais pas les commentaires.

4.3.3.3 Référence d'URI same-document

Déréférencer une référence same-document DOIT résulter en un ensemble-nœud XPath convenable pour être utilisé par le XML canonique [RFC3076]. Précisément, déréférencer un URI nul (URI="") DOIT résulter en un ensemble-nœud XPath qui inclut tout nœud non commentaire du document XML contenant l'attribut URI. Dans un fragment d'URI, les caractères après le caractère numéro ('#') se conforment à la syntaxe de [XPointer]. Lorsque on traite un XPointer, l'application DOIT se comporter comme si le nœud racine du document XML contenant l'attribut d'URI était utilisé pour initialiser le contexte d'évaluation XPointer. L'application DOIT se comporter comme si le résultat du traitement de XPointer était un ensemble-nœud déduit de l'ensemble de localisation résultant comme suit :

1. éliminer les nœuds point
2. remplacer chaque nœud de gamme par tous les nœuds XPath qui ont un contenu complet ou partiel au sein de la gamme
3. remplacer le nœud racine par ses enfants (si ils sont dans l'ensemble-nœud)
4. remplacer tout élément nœud E par E plus tous les descendants de E (texte, commentaire, PI, élément) et tous les nœuds d'espace de noms et d'attribut de E et ses éléments descendants
5. si l'URI n'est pas un Xpointer complet, supprimer alors tous les nœuds commentaire.

Les remplacements du second au dernier sont nécessaires parce que XPointer indique normalement une sous-arborescence de l'arborescence d'analyse de document XML qui utilise juste le nœud d'élément qui est à la racine de la sous-arborescence, tandis que XML canonique traite un ensemble-nœud comme un ensemble de nœuds dans lequel l'absence de nœuds descendants résulte en l'absence de leur texte représentatif à partir de la forme canonique.

La dernière étape est effectuée pour les URI nuls, les XPointers de nom nu et les Xpointers de séquence fille. Elle est nécessaire parce que lorsque la [RFC3076] passe un ensemble-nœud, elle traite l'ensemble-nœud comme il est : avec ou sans commentaire. C'est seulement lorsque elle est invoquée avec un flux d'octets qu'elle invoque ses propres expressions Xpath (par défaut ou sans commentaire). Pour retenir donc le comportement par défaut d'élimination des commentaires lorsque passe un ensemble-nœud, ils sont retirés dans la dernière étape si l'URI n'est pas un XPointer complet. Pour garder les commentaires lors du choix d'un élément par un identifiant, utiliser le XPointer complet suivant :

URI='#xpointer(id('ID'))'.

Pour garder les commentaires lors du choix du document entier, utiliser le XPointer complet suivant :

URI='#xpointer(/)'.

Cet XPointer contient une expression XPath simple qui inclut le nœud racine, que les étapes de la seconde à la dernière ci-dessus remplacent par tous les nœuds de l'arborescence d'analyse (tous les descendants, plus tous les attributs, plus tous les nœuds d'espaces de noms).

4.3.3.4 Élément Transforms

L'élément facultatif Transforms contient une liste ordonnée d'élements Transform ; ils décrivent comment le signataire a obtenu l'objet de données qui a été résumé. Le résultat de chaque Transform sert d'entrée à la prochaine Transform. L'entrée de la première Transform est le résultat du déréférencement de l'attribut URI de l'élément Reference. Le résultat de la dernière Transform est l'entrée pour l'algorithme DigestMethod. Lorsque les transformations sont appliquées, le signataire ne signe pas le document natif (original) mais le document résultant (transformé). (Voir au paragraphe 8.1, Seul ce qui est signé est sûr.)

Chaque transformation consiste en paramètres d'attribut et de contenu d'algorithme, si il en est, appropriés pour l'algorithme donné. La valeur de l'attribut Algorithm spécifie le nom de l'algorithme à effectuer, et le contenu de la transformation fournit des données supplémentaires pour diriger le traitement par l'algorithme des entrées à la transformation. (Voir à la section 6 "Identifiants d'algorithme et exigences de mise en œuvre".)

Comme décrit dans le modèle de traitement de référence (paragraphe 4.3.3.2) certaines transformations prennent en entrée un ensemble-nœud XPath, alors que d'autres exigent un flux d'octets. Si l'entrée réelle correspond aux besoins d'entrées de la transformation, celle-ci s'opère alors sur l'entrée non altérée. Si les exigences d'entrées de la transformation diffèrent du format de l'entrée réelle, l'entrée doit alors être convertie.

Certaines transformations peuvent exiger un type MIME explicite, un jeu de caractères ("jeu de caractères" enregistré auprès de l'IANA) ou autres informations de ce genre concernant les données qui sont reçues d'une transformation antérieure ou de la source des données, bien qu'aucun algorithme de transformation spécifié dans le présent document n'ait besoin de telles informations explicites. De telles caractéristiques de données sont fournies comme paramètres à l'algorithme de transformation et devraient être décrites dans la spécification de l'algorithme.
Parmi les exemples de transformations, on citera, entre autres le décodage base64 [RFC2045], la canonisation [RFC3076], le filtrage XPath [XPath], et XSLT [XSLT]. La définition générique de l'élément Transform permet aussi des algorithmes de transformation spécifiques de l'application. Par exemple, la transformation pourrait être un sous-programme de décompression donné par une classe Java apparaissant comme un paramètre codé en base64 sur un algorithme de transformation Java. Cependant, les applications devraient s'interdire d'utiliser des transformations spécifiques de l'application si elles souhaitent que leurs signatures soient vérifiables en-dehors de leur domaine d'application. Le paragraphe 6.6, Algorithmes de transformation, définit la liste des transformations standard.

Définition du schéma :

```
<nom d'élément="Transforms" type="ds:TransformsType"/>
<nom de complexType ="TransformsType">
  <sequence>
    <element ref="ds:Transform" maxOccurs=«non limité»/>
  </sequence>
</complexType>

<nom d'élément="Transform" type="ds:TransformType"/>
<nom de complexType ="TransformType" mixte="vrai">
  <choix minOccurs="0" maxOccurs=«non limité»>
    <tout espace de noms="##autre" processContents="lâche"/>
    <!-- (1,1) éléments de (0,nonlimité) espaces de noms -->
    <nom d'élément="XPath" type="chaîne"/>
  </choix>
  <nom d'attribut="Algorithm" type="anyURI" usage="exigé"/>
</complexType>
```

DTD :

```
<!ELEMENT Transforms (Transform+)>

<!ELEMENT Transform (#PCDATA|XPath %Transform.ANY;)* >
<!ATTLIST Transform
 Algorithm    CDATA    #EXIGÉ >
```

```
<!ELEMENT XPath (#PCDATA) >
```

4.3.3.5 Élément DigestMethod

DigestMethod *(méthode de résumé)* est un élément exigé qui identifie l'algorithme de résumé à appliquer à l'objet signé. Cet élément utilise la structure générale donnée ici pour les algorithmes spécifiés au paragraphe 6.1 "Identifiants d'algorithme et exigences de mise en œuvre".

Si le résultat du déréférencement de l'URI et l'application de Transforms est un ensemble-nœud XPath (ou un remplacement suffisamment fonctionnel mis en œuvre par l'application) il doit alors être converti comme décrit dans le modèle de traitement de référence (paragraphe 4.3.3.2). Si le résultat du déréférencement d'URI et l'application de la transformation est un flux d'octets, aucune conversion ne survient alors (il peut y avoir des commentaires si XML canonique avec commentaires était spécifié dans le Transforms). L'algorithme de résumé est appliqué aux octets de données du flux d'octets résultant.

Définition du schéma :

```
<nom d'élément="DigestMethod" type="ds:DigestMethodType"/>
<complexType name="DigestMethodType" mixte="vrai">
  <sequence>
    <tout espace de noms="##autre" processContents="lâche"
        minOccurs="0" maxOccurs="non limité"/>
  </sequence>
  <nom d'attribut="Algorithm" type="anyURI" usage="exigé"/>
</complexType>
```

DTD :

```
<!ELEMENT DigestMethod (#PCDATA %Method.ANY;)* >
<!ATTLIST DigestMethod
 Algorithm      CDATA   #EXIGÉ >
```

4.3.3.6 Élément DigestValue

DigestValue *(valeur de résumé)* est un élément qui contient la valeur codée du résumé. Le résumé est toujours codé en utiilisant le base64 [RFC2045].

Définition du schéma :

```
<nom d'élément="DigestValue" type="ds:DigestValueType"/>
<nom simpleType ="DigestValueType">
  <restriction base="base64Binary"/>
</simpleType>
```

DTD :

```
<!ELEMENT DigestValue  (#PCDATA)  >
<!-- valeur de résumé codée en base64 -->
```

4.4 Élément KeyInfo

KeyInfo *(informations clés)* est un élément facultatif qui permet au ou aux receveurs d'obtenir la clé nécessaire pour valider la signature. KeyInfo peut contenir des clés, des noms, des certificats et d'autres informations de gestion de clé publique, telles que des données de distribution de clé dans la bande ou d'accord de clé. La présente spécification définit quelques types simples mais les applications peuvent étendre ces types ou les remplacer tous par leur propre sémantique d'identification et d'échange de clés en utilisant la facilité d'expace de noms XML de [XML-ns]. Cependant, les questions de confiance qu'on peut accorder à de telles informations de clés (par exemple, leur force ou leur authenticité) sortent du domaine d'application de la présente spécification et relèvent de l'application.

Si KeyInfo est omis, le receveur est supposé être capable d'identifier la clé sur la base du contexte de l'application. Plusieurs déclarations au sein de KeyInfo se réfèrent à la même clé. Bien que les applications puissent définir et utiliser tout mécanisme de leur choix par l'inclusion d'éléments provenant de différents espaces de noms, les versions conformes DOIVENT mettre en œuvre KeyValue (paragraphe 4.4.2) et DEVRAIENT mettre en œuvre RetrievalMethod (paragraphe 4.4.3).

Les spécifications de schéma/DTD de beaucoup des enfants de KeyInfo (par exemple, PGPData, SPKIData, X509Data) permettent d'étendre/compléter leur contenu avec des éléments provenant d'un autre espace de noms. Cela ne peut être fait que si il est sûr d'ignorer ces éléments d'extension tout en revendiquant la prise en charge des types définis dans la présente spécification. Autrement, les éléments externes, y compris des structures de substitution à celles définies par la présente spécification, DOIVENT être des descendants de KeyInfo. Par exemple, si est définie une norme XML-PGP complète, son élément racine DEVRA être un enfant de KeyInfo. (Bien sûr, de nouvelles structures provenant d'espaces de noms externes peuvent incorporer des éléments provenant de l'espace de noms &dsig; via des caractéristiques du langage de définition du type. Par exemple, elles peuvent créer un DTD qui mélange leurs propres éléments qualifiés et ceux de dsig, ou un schéma qui permet, inclus, importe, ou déduit de nouveaux types fondés sur des éléments &dsig;.)

La liste suivante récapitule les types de KeyInfo qui sont alloués à un identifiant dans l'espace de noms &dsig; ; ils peuvent être utilisés au sein du type d'attribut RetrievalMethod pour décrire une structure KeyInfo distante.

* http://www.w3.org/2000/09/xmldsig#DSAKeyValue
* http://www.w3.org/2000/09/xmldsig#RSAKeyValue
* http://www.w3.org/2000/09/xmldsig#X509Data
* http://www.w3.org/2000/09/xmldsig#PGPData
* http://www.w3.org/2000/09/xmldsig#SPKIData
* http://www.w3.org/2000/09/xmldsig#MgmtData

En plus des types ci-dessus pour lesquels on définit une structure XML, on spécifie un type supplémentaire pour indiquer un certificat binaire (ASN.1 DER) X.509.

* http://www.w3.org/2000/09/xmldsig#rawX509Certificate

Définition du schéma : <nom d'élément="KeyInfo" type="ds:KeyInfoType"/>

```
<nom de complexType="KeyInfoType" mixte="vrai">
 <choix maxOccurs="non limité">
  <element ref="ds:KeyName"/>
  <element ref="ds:KeyValue"/>
  <element ref="ds:RetrievalMethod"/>
  <element ref="ds:X509Data"/>
  <element ref="ds:PGPData"/>
  <element ref="ds:SPKIData"/>
  <element ref="ds:MgmtData"/>
  <tout processContents="lax" espace de nom ="##autre"/>
  <!-- (1,1) éléments provenant des espaces de nom (0,non limité) -->
 </choix>
 <nom d'attribut="Id" type="ID" usage="optionnel"/>
</complexType>
```

DTD :

```
<!ELEMENT KeyInfo (#PCDATA|KeyName|KeyValue|RetrievalMethod|
        X509Data|PGPData|SPKIData|MgmtData %KeyInfo.ANY;)* >
<!ATTLIST KeyInfo
 Id  ID   #IMPLICITE >
```

4.4.1 Élément KeyName

L'élément KeyName *(nom de clé)* contient une valeur de chaîne (dans laquelle les espaces sont significatives) qui peut être utilisée par le signataire pour communiquer un identifiant de clé au receveur. Normalement, KeyName contient un identifiant qui se rapporte à la paire de clés utilisée pour signer le message, mais il peut contenir d'autres informations en rapport avec le protocole qui identifient indirectement une paire de clés. (Les utilisations courantes de KeyName incluent de simples noms de chaîne pour des clés, un indice de clés, un nom distinctif (DN), une adresse de messagerie électronique, etc.)

Définition du schéma : <nom d'élément="KeyName" type="chaîne"/>

DTD : <!ELEMENT KeyName (#PCDATA) >

4.4.2 Élément KeyValue

L'élément KeyValue *(valeur de clé)* contient une seule clé publique qui peut être utile pour valider la signature. Les

formats structurés pour définir les clés publiques DSA (EXIGÉ) et RSA (RECOMMANDÉ) sont définis au paragraphe 6.4 "Algorithmes de signature". L'élément KeyValue peut inclure des valeurs de clé publique définies en externe représentées comme PCDATA ou des types d'éléments provenant d'un espace de noms externe.

Définition du schéma :

```
<nom d'élément="KeyValue" type="ds:KeyValueType"/>
<nom complexType ="KeyValueType" mixte="vrai">
 <choix>
  <element ref="ds:DSAKeyValue"/>
  <element ref="ds:RSAKeyValue"/>
  <tout espace de noms ="##autre" processContents="lâche"/>
 </choix>
</complexType>
```

DTD : <!ELEMENT KeyValue (#PCDATA|DSAKeyValue|RSAKeyValue %KeyValue.ANY;)* >

4.4.2.1 Élément DSAKeyValue

Identifiant
Type="http://www.w3.org/2000/09/xmldsig#DSAKeyValue" (cela peut être utilisé au sein d'un élément RetrievalMethod ou Reference pour identifier le type du référent).

Les clés DSA et l'algorithme de signature DSA sont spécifiés dans [DSS]. Les valeurs de clé publique DSA ont les champs suivants :

P :	modulo un nombre premier satisfaisant aux exigences de [DSS].
Q :	un entier dans la gamme 2**159 < Q < 2**160 qui est un nombre premier diviseur de P-1.
G :	un entier avec certaines propriétés par rapport à P et Q.
Y :	G**X mod P (où X fait partie de la clé privée et n'est pas rendue publique).
J :	(P - 1) / Q.
seed :	un germe de génération de nombre premier DSA.
pgenCounter :	un compteur de génération de nombre premier DSA.

Le paramètre J n'est proposé à l'inclusion que pour des raisons d'efficacité car il est calculable à partir de P et Q. Les paramètres seed et pgenCounter sont utilisés dans l'algorithme DSA de génération de nombres premiers spécifié dans [DSS]. À ce titre, ils sont facultatifs, mais doivent être tous deux présents ou absents. Cet algorithme de génération de nombres premiers a été conçu pour fournir l'assurance qu'un premier faible n'est pas utilisé et qu'il donne une valeur de P et de Q. Les paramètres P, Q, et G peuvent être publics et communs à un groupe d'utilisateurs. Ils peuvent être connus à partir du contexte d'application. À ce titre, ils sont facultatifs mais P et Q doivent tous deux apparaître ou tous deux être absents. Si P, Q, seed, et pgenCounter sont tous présents, les mises en œuvre ne sont pas obligées de vérifier si ils sont cohérents et elles ont toute liberté pour utiliser P et Q ou seed et pgenCounter. Tous les paramètres sont codés en valeurs de base64 [RFC2045].

Des entiers de longueur arbitraire (par exemple, "bignums" comme des modulo deRSA) sont représentés en XML comme des chaînes d'octets, comme défini par le type ds:CryptoBinary.

Définition du schéma :

```
<nom d'élément="DSAKeyValue" type="ds:DSAKeyValueType"/>
<nom de complexType="DSAKeyValueType">
 <sequence>
  <sequence minOccurs="0">
   <nom d'élément="P" type="ds:CryptoBinary"/>
   <nom d'élément="Q" type="ds:CryptoBinary"/>
  </sequence>
  <nom d'élément="G" type="ds:CryptoBinary" minOccurs="0"/>
  <nom d'élément="Y" type="ds:CryptoBinary"/>
  <nom d'élément="J" type="ds:CryptoBinary" minOccurs="0"/>
  <sequence minOccurs="0">
   <nom d'élément="Seed" type="ds:CryptoBinary"/>
   <nom d'élément="PgenCounter" type="ds:CryptoBinary"/>
  </sequence>
 </sequence>
</complexType>
```

Définition DTD :

```
<!ELEMENT DSAKeyValue ((P, Q)?, G?, Y, J?, (Seed, PgenCounter)?) >
<!ELEMENT P (#PCDATA) >
<!ELEMENT Q (#PCDATA) >
<!ELEMENT G (#PCDATA) >
<!ELEMENT Y (#PCDATA) >
<!ELEMENT J (#PCDATA) >
<!ELEMENT Seed (#PCDATA) >
<!ELEMENT PgenCounter (#PCDATA) >
```

4.4.2.2 Élément RSAKeyValue

Identifiant :
Type="http://www.w3.org/2000/09/xmldsig#RSAKeyValue" (cela peut être utilisé au sein d'un élément RetrievalMethod ou Reference pour identifier le type de référent)

Les valeurs de clé RSA ont deux champs : Modulus et Exponent.

```
<RSAKeyValue>
  <Modulus>
   xA7SEU+e0yQH5rm9kbCDN9o3aPIo7HbP7tX6WOocLZAtNfyxSZDU16ksL6W
   jubafOqNEpcwR3RdFsT7bCqnXPBe5ELh5u4VEy19MzxkXRgrMvavzyBpVRg
   BUwUlV5foK5hhmbktQhyNdy/6LpQRhDUDsTvK+g9Ucj47es9AQJ3U=
  </Modulus>
  <Exponent>AQAB</Exponent>
</RSAKeyValue>
```

Des entiers de longueur arbitraire (par exemple, "bignums" comme des modulo deRSA) sont représentés en XML comme des chaînes d'octets, comme défini par le type ds:CryptoBinary.

Définition du schéma :

```
<nom d'élément="RSAKeyValue" type="ds:RSAKeyValueType"/>
<nom de complexType="RSAKeyValueType">
  <sequence>
    <nom d'élément="Modulus" type="ds:CryptoBinary"/>
    <nom d'élément="Exponent" type="ds:CryptoBinary"/>
  </sequence>
</complexType>
```

Définition DTD :

```
<!ELEMENT RSAKeyValue (Modulus, Exponent) >
<!ELEMENT Modulus (#PCDATA) >
<!ELEMENT Exponent (#PCDATA) >
```

4.4.3 Élément RetrievalMethod

Un élément RetrievalMethod *(méthode de restitution)* au sein de KeyInfo est utilisé pour porter une référence aux informations KeyInfo qui sont mémorisées dans un autre lieu. Par exemple, plusieurs signatures dans un document peuvent utiliser une clé vérifiée par une chaîne de certificats X.509v3 qui apparaît une fois dans le document, ou à distance, en dehors du document ; le KeyInfo de chaque signature peut faire référence à cette chaîne en utilisant un seul élément RetrievalMethod au lieu d'inclure la chaîne entière avec une séquence d'éléments X509Certificate.

RetrievalMethod utilise la même syntaxe et le même comportement de déréférencement que l'URI de Reference (paragraphe 4.3.3.1) et que le modèle de traitement de Reference (paragraphe 4.3.3.2) sauf qu'il n'y a pas d'élément fils de DigestMethod ou de DigestValue et que la présence de l'URI est obligatoire.

Type est un identifiant facultatif pour le type de données à restituer. Le résultat du déréférencement d'une référence RetrievalMethod pour tous les types KeyInfo définis par la présente spécification (paragraphe 4.4) avec une structure XML correspondante est un élément ou document XML avec cet élément comme racine. Le KeyInfo rawX509Certificate (pour lequel il n'y a pas de structure XML) retourne un certificat X509 binaire.

Définition du schéma :

```
<nom d'élément="RetrievalMethod" type="ds:RetrievalMethodType"/>
<nom de complexType="RetrievalMethodType">
 <sequence>
  <element ref="ds:Transforms" minOccurs="0"/>
 </sequence>
 <nom d'attribut="URI" type="anyURI"/> usage="exigé"/>
 <nom d'attribut="Type" type="anyURI" usage="optionnel"/>
</complexType>

DTD:
<!ELEMENT RetrievalMethod (Transforms?) >
<!ATTLIST RetrievalMethod
  URI   CDATA #EXIGÉ
  Type  CDATA #IMPLICITE >
```

4.4.4 Élément X509Data

Identifiant
Type="http://www.w3.org/2000/09/xmldsig#X509Data" (cela peut être utilisé au sein d'un élément RetrievalMethod ou Reference pour identifier le type du référent).

Un élément X509Data au sein de KeyInfo contient un ou plusieurs identifiants de clés ou certificats X509 (ou d'identifiants de certificats ou d'une liste de révocation). Le contenu de X509Data est :

- Au moins un élément, tiré de l'ensemble suivant de types d'éléments ; chacun d'eux peut apparaître avec d'autres ou plus d'une fois si, et seulement si chaque instance décrit le, ou se rapporte au, même certificat :
  - o l'élément X509IssuerSerial qui contient une paire numéro distinctif/numéro de série de producteur X.509 qui DEVRAIT être conforme à la [RFC2253],
  - o l'élément X509SubjectName qui contient un nom distinctif de sujet X.509 qui DEVRAIT être conforme à la [RFC2253],
  - o l'élément X509SKI qui contient la valeur codée en base64 plein (c'est-à-dire, non codé en DER) d'une extension X509 V.3 SubjectKeyIdentifier,
  - o l'élément X509Certificate qui contient un certificat codé en base64 [X509v3],
  - o des éléments provenant d'un espace de nom externe qui accompagnent/complètent tout élément ci-dessus,
  - o l'élément X509CRL qui contient une liste de révocation de certificats (CRL) codée en base64 [X509v3].

Tout élément X509IssuerSerial, X509SKI, et X509SubjectName qui apparaît DOIT se référer au ou aux certificats qui contiennent la clé de validation. Tous les éléments qui se réfèrent à un certificat individuel particulier DOIVENT être groupés à l'intérieur d'un seul élément X509Data et si le certificat auquel ils se réfèrent apparaît, il DOIT aussi être dans cet élément X509Data.

Tous les éléments X509IssuerSerial, X509SKI, et X509SubjectName qui se rapportent à la même clé mais à des certificats différents DOIVENT être regroupés au sein d'un seul KeyInfo mais PEUVENT survenir dans plusieurs éléments X509Data.

Tous les certificats qui apparaissent dans un élément X509Data DOIVENT se rapporter à la clé de validation soit en la contenant, soit en faisant partie de la chaîne de certification qui se termine dans un certificat contenant la clé de validation.

Aucun ordre n'est impliqué par les contraintes ci-dessus. Les commentaires donnés dans l'instance suivante montrent ces contraintes :

```
<KeyInfo>
 <X509Data> <!                          -- deux pointeurs sur le certificat -A -->
  <X509IssuerSerial>
   <X509IssuerName>CN=TAMURA Kent, OU=TRL, O=IBM,
    L=Yamato-shi, ST=Kanagawa, C=JP</X509IssuerName>
   <X509SerialNumber>12345678</X509SerialNumber>
  </X509IssuerSerial>
  <X509SKI>31d97bd7</X509SKI>
 </X509Data>
 <X509Data><!                           -- un seul pointeur sur le certificat -B -->
  <X509SubjectName>Sujet du certificat B</X509SubjectName>
```

```
  </X509Data>
  <X509Data> <!                          -- chaîne de certificats -->
   <!--Signer cert, issuer CN=arbolCA,OU=FVT,O=IBM,C=US, serial 4-->
   <X509Certificate>MIICXTCCA..</X509Certificate>
   <!—Sujet du certificat intermédiaire CN=arbolCA,OU=FVT,O=IBM,C=US
       issuer CN=tootiseCA,OU=FVT,O=Bridgepoint,C=US -->
   <X509Certificate>MIICPzCCA...</X509Certificate>
   <!-- Root cert subject CN=tootiseCA,OU=FVT,O=Bridgepoint,C=US -->
   <X509Certificate>MIICSTCCA...</X509Certificate>
  </X509Data>
 </KeyInfo>
```

Noter qu'il n'y a pas de disposition directe pour un "paquet" codé PKCS#7 de certificats ou de CRL. Cependant, un ensemble de certificats et de CRL peut se produire au sein d'un élément X509Data et plusieurs éléments X509Data peuvent survenir dans un KeyInfo. Chaque fois que plusieurs certificats surviennent dans un élément X509Data, au moins un de ces certificats doit contenir la clé publique qui vérifie la signature.

Aussi, les chaînes dans DNames (X509IssuerSerial, X509SubjectName, et KeyName si approprié) devraient être codées comme suit :

* Considérer la chaîne comme composée de caractères Unicode.
* Esquiver les occurrences des caractères spéciaux suivants en les faisant précéder du caractère "\" : un caractère "#" survenant au début de la chaîne ou un des caractères ",", "+", """, "\", "<", ">" ou ";".
* Esquiver toutes les occurrences de caractères de contrôle ASCII (gamme Unicode de \x00 à \x 1f) en les remplaçant par "\" suivi par le nombre hexadécimal de deux chiffres représentant son numéro Unicode.
* Esquiver toute espace blanche en queue en remplaçant "\ " par "\20".
* Comme un document XML consiste logiquement en caractères, et non en octets, la chaîne Unicode résultante est finalement codée conformément au codage de caractères utilisé pour produire la représentation physique du document XML.

Définition du schéma :

```
<nom d'élément="X509Data" type="ds:X509DataType"/>
<nom de complexType="X509DataType">
 <sequence maxOccurs=«non limité»>
  <choix>
   <nom d'élément="X509IssuerSerial"
         type="ds:X509IssuerSerialType"/>
   <nom d'élément="X509SKI" type="base64Binary"/>
   <nom d'élément="X509SubjectName" type=«chaîne»/>
   <nom d'élément="X509Certificate" type="base64Binary"/>
   <nom d'élément="X509CRL" type="base64Binary"/>
   <tout espace de noms="##autre" processContents="lâche"/>
  </choix>
 </sequence>
</complexType>
<complexType name="X509IssuerSerialType">
 <sequence>
  <nom d'élément="X509IssuerName" type=«chaîne»/>
  <nom d'élément="X509SerialNumber" type=«entier»/>
 </sequence>
</complexType>
```

DTD :

```
<!ELEMENT X509Data ((X509IssuerSerial | X509SKI | X509SubjectName
            | X509Certificate | X509CRL)+ %X509.ANY;)>
<!ELEMENT X509IssuerSerial (X509IssuerName, X509SerialNumber) >
<!ELEMENT X509IssuerName (#PCDATA) >
<!ELEMENT X509SubjectName (#PCDATA) >
<!ELEMENT X509SerialNumber (#PCDATA) >
<!ELEMENT X509SKI (#PCDATA) >
<!ELEMENT X509Certificate (#PCDATA) >
<!ELEMENT X509CRL (#PCDATA) >
```

<!-- Note, ce DTD et ce schéma permettent que X509Data soit vide ; cela est exclu par le texte de l'élément KeyInfo

(paragraphe 4.4) qui déclare qu'au moins un élément provenant de l'espace de noms dsig devrait être présent dans les structures PGP, SPKI, et X509. Cela s'exprime facilement pour les autres types de clé, mais pas pour X509Data à cause de sa riche structure. -->

4.4.5 Élément PGPData

Identifiant :
Type="http://www.w3.org/2000/09/xmldsig#PGPData" (cela peut être utilisé au sein d'un élément RetrievalMethod ou Reference pour identifier le type du référent).

L'élément PGPData au sein de KeyInfo est utilisé pour porter les informations qui se rapportent aux paires de clé publiques PGP et aux signatures sur de telles clés. La valeur de PGPKeyID est une séquence base64Binary qui contient un identifiant de clé publique PGP standard comme défini au paragraphe 11.2 de la [RFC2440]. Le PGPKeyPacket contient un paquet de matériel de clé codé en base64 comme défini au paragraphe 5.5 de la [RFC2440]. Ces types d'élément fils peuvent être complétés/étendus par des frères provenant d'un espace de nom externe au sein de PGPData, ou PGPData peut être remplacé tout entier par une autre structure PGP XML comme enfant de KeyInfo. PGPData doit contenir un PGPKeyID et/ou un PGPKeyPacket et 0, un ou plusieurs éléments provenant d'un espace de noms externe.

Définition du schéma :

```
<nom d'élément="PGPData" type="ds:PGPDataType"/>
<nom de complexType="PGPDataType">
 <choix>
  <sequence>
   <nom d'élément="PGPKeyID" type="base64Binary"/>
   <nom d'élément="PGPKeyPacket" type="base64Binary"
         minOccurs="0"/>
   <tout espace de nom="##autre" processContents="lâche" minOccurs="0"
    maxOccurs=«non limité»/>
  </sequence>
  <sequence>
   <nom d'élément="PGPKeyPacket" type="base64Binary"/>
   <tout espace de nom="##autre" processContents="lâche" minOccurs="0"
    maxOccurs=«non limité»/>
  </sequence>
 </choix>
</complexType>
```

DTD :

```
<!ELEMENT PGPData ((PGPKeyID, PGPKeyPacket?) | (PGPKeyPacket)
          %PGPData.ANY;) >
<!ELEMENT PGPKeyPacket  (#PCDATA)  >
<!ELEMENT PGPKeyID  (#PCDATA)  >
```

4.4.6 Élément SPKIData

Identifiant :
Type="http://www.w3.org/2000/09/xmldsig#SPKIData" (cela peut être utilisé au sein d'un élément RetrievalMethod ou Reference pour identifier le type du référent).

L'élément SPKIData au sein de KeyInfo est utilisé pour porter des informations qui se rapportent aux paires de clé publique SPKI, aux certificats et autres données SPKI. SPKISexp est le codage en base64 d'une expression S canonique SPKI. SPKIData doit avoir au moins une SPKISexp ; SPKISexp peut être complété/étendu par un frère provenant d'un espace de noms externe au sein de SPKIData, ou SPKIData peut être entièrement remplacé par une autre structure SPKI XML comme enfant de KeyInfo.

Définition du schéma :

```
<nom d'élément="SPKIData" type="ds:SPKIDataType"/>
<nom de complexType="SPKIDataType">
 <sequence maxOccurs=«non limité»>
  <nom d'élément="SPKISexp" type="base64Binary"/>
  <tout espace de noms="##autre" processContents="lâche" minOccurs="0"/>
```

```
  </sequence>
 </complexType>
```

DTD :

```
 <!ELEMENT SPKIData (SPKISexp %SPKIData.ANY;) >
 <!ELEMENT SPKISexp  (#PCDATA) >
```

4.4.7 Élément MgmtData

Identifiant :
Type="http://www.w3.org/2000/09/xmldsig#MgmtData" (cela peut être utilisé au sein d'un élément RetrievalMethod ou Reference pour identifier le type du référent).

L'élément MgmtData au sein de KeyInfo est une valeur de chaîne utilisée pour porter des données de distribution ou d'accord de clé dans la bande. Par exemple, d'échange de clé DH, de chiffrement de clé RSA, etc. L'utilisation de cet élément est NON RECOMMANDÉ. Il fournit une accroche syntaxique où peuvent être placées des données de distribution ou d'accord de clé dans la bande. Cependant, des éléments fils supérieurs interopérables de KeyInfo pour la transmission de clés chiffrées et d'accord de clé sont en cours de spécification par le groupe de travail Chiffrement XML du W3C et ils devraient être utilisés plutôt que MgmtData.

Définition du schéma : <nom d'élément="MgmtData" type=«chaîne»/>

DTD : <!ELEMENT MgmtData (#PCDATA)>

4.5 Élément Object

Identifiant :
Type="http://www.w3.org/2000/09/xmldsig#Object" (cela peut être utilisé au sein d'un élément Reference pour identifier le type du référent).

Object est un élément facultatif qui peut survenir une ou plusieurs fois. Lorsque il est présent, cet élément peut contenir n'importe quelles données. L'élément Object peut inclure un type MIME facultatif, un identifiant, et des attributs de codage.

Le codage attribué à Object peut être utilisé pour fournir un URI qui identifie la méthode par laquelle l'objet est codé (par exemple, un fichier binaire).

Le type Mimeattribut est un attribut facultatif qui décrit les données au sein de Object (indépendamment de son codage). C'est une chaîne avec des valeurs définies par la [RFC2045]. Par exemple, si Object contient un PNG codé en base64, le codage peut être spécifié comme 'http://www.w3.org/2000/09/xmldsig#base64' et le type Mime comme 'image/png'. Cet attribut est purement indicatif ; aucune validation des informations du MimeType n'est exigée par la présente spécification. Les applications qui requièrent des informations de type normatif et de codage pour la validation de signature devraient spécifier les transformations avec des types et/ou codages résultants bien définis.

L'identifiant de Object est normalement référencé à partir d'une Reference dans SignedInfo, ou Manifest. Cet élément est normalement utilisé pour des signatures enveloppantes où l'objet à signer est à inclure dans l'élément de signature. Le résumé est calculé sur l'élément Object entier incluant les étiquettes de début et de fin.

Noter que si l'application souhaite exclure les étiquettes <Object> du calcul du résumé, la référence doit identifier l'objet de données réel (ce qui est facile pour les documents XML) ou une transformation doit être utilisée pour retirer les étiquettes de Object (vraisemblablement lorsque l'objet de données est non XML). L'exclusion des étiquettes de l'objet peut être souhaitée dans des cas où on veut que la signature reste valide si l'objet de données est déplacé de l'intérieur à l'extérieur d'une signature (ou vice-versa) ou lorsque le contenu de l'objet est un codage d'un document original binaire et qu'on désire extraire et décoder de façon à signer la représentation originale au bit près.

Définition du schéma :

```
    <nom d'élément="Object" type="ds:ObjectType"/>
    <nom de complexType="ObjectType" mixte="vrai">
     <sequence minOccurs="0" maxOccurs=«non limité»>
      <tout espace de noms="##tout" processContents="lâche"/>
```

```
    </sequence>
    <nom d'attribut="Id" type="ID" use="optionnel"/>
    <nom d'attribut="MimeType" type=«chaîne» use="optionnel"/>
    <nom d'attribut="Encoding" type="anyURI" usage="optionnel"/>
  </complexType>
```

DTD :

```
  <!ELEMENT Object (#PCDATA|Signature|SignatureProperties|Manifest
            %Object.ANY;)* >
  <!ATTLIST Object
   Id  ID  #IMPLICITE
   MimeType    CDATA   #IMPLICITE
   Encoding    CDATA   #IMPLICITE >
```

5. Syntaxe additionnelle de Signature

Cette section décrit les éléments de mise en œuvre facultative Manifest et SignatureProperties et décrit le traitement des instructions et commentaires du processus XML. Pour ce qui concerne les éléments Manifest et SignatureProperties, cette section spécifie la syntaxe et un peu du comportement – cela est du domaine de l'application. Ces éléments peuvent apparaître partout où le permet le modèle de contenu parent ; le modèle de contenu Signature ne les permet qu'au sein de Object.

5.1 Élément Manifest

Identifiant :
Type="http://www.w3.org/2000/09/xmldsig#Manifest" (cela peut être utilisé au sein d'un élément Reference pour identifier le type du référent).

L'élément Manifest fournit une liste des références. La différence avec la liste de SignedInfo est qu'il est défini, par l'application, lequel des résumés, s'il en est, est en fait vérifié par rapport aux objets référencés, et ce qu'il convient de faire si l'objet est inaccessible ou si la comparaison de résumés échoue. Si un Manifest est pointé par les SignedInfo, le résumé sur le Manifest lui-même sera vérifié par le comportement central de validation de signature. Les résumés au sein d'un tel Manifest sont vérifiés à la discrétion de l'application. Si un Manifest est référencé à partir d'un autre Manifest, même le résumé global de ce Manifest à deux niveaux de profondeur pourrait n'être pas vérifié.

Définition du schéma :

```
  <nom d'élément="Manifest" type="ds:ManifestType"/>
  <nom complexType="ManifestType">
    <sequence>
      <element ref="ds:Reference" maxOccurs=«non limité»/>
    </sequence>
    <nom d'attribut="Id" type="ID" usage="optionnel"/>
  </complexType>
```

DTD :

```
  <!ELEMENT Manifest (Reference+)  >
  <!ATTLIST Manifest
        Id ID  #IMPLICITE >
```

5.2 Élément SignatureProperties

Identifiant :
Type="http://www.w3.org/2000/09/xmldsig#SignatureProperties" (cela peut être utilisé au sein d'un élément Reference pour identifier le type du référent).

Des éléments d'information supplémentaires concernant la génération de la ou des signatures peuvent être placés dans un élément SignatureProperty (c'est-à-dire, un horodatage, ou le numéro de série du matériel de chiffrement utilisé pour la génération de la signature).

Définition du schéma :

```
    <nom d'élément="SignatureProperties"
          type="ds:SignaturePropertiesType"/>
    <complexType name="SignaturePropertiesType">
      <sequence>
        <element ref="ds:SignatureProperty" maxOccurs=«non limité»/>
      </sequence>
      <nom d'attribut="Id" type="ID" usage="optionnel"/>
    </complexType>

    <nom d'élément="SignatureProperty"
          type="ds:SignaturePropertyType"/>
    <complexType name="SignaturePropertyType" mixte="vrai">
      <choix maxOccurs=«non limité»>
        <tout espace de noms="##autrr" processContents="lâche"/>
        <!-- (1,1) éléments provenant d'espaces de noms (1, non limité) -->
      </choix>
      <nom d'attribut="Target" type="anyURI" usage="exigé"/>
      <nom d'attribut="Id" type="ID" usage="optionnel"/>
    </complexType>
```

DTD :

```
    <!ELEMENT SignatureProperties (SignatureProperty+)  >
    <!ATTLIST SignatureProperties
          Id    ID     #IMPLICITE >

    <!ELEMENT SignatureProperty (#PCDATA %SignatureProperty.ANY;)* >
    <!ATTLIST SignatureProperty
          Target CDATA   #EXIGÉ
          Id    ID     #IMPLICITE  >
```

5.3 Instructions de traitement dans les éléments Signature

La présente spécification n'utilise pas d'instruction de traitement (PI, *processing instruction*) XML.

Noter que les PI placés à l'intérieur de SignedInfo par une application seront signés sauf si l'algorithme CanonicalizationMethod les élimine. (Ceci est vrai pour tout contenu XML signé.) Toutes les CanonicalizationMethod identifiées dans la présente spécification conservent les PI. Lorsque un PI fait partie d'un contenu qui est signé (par exemple, au sein de SignedInfo ou de documents XML référencés) tout changement du PI va évidemment résulter en un échec de la signature.

5.4 Commentaires dans les éléments Signature

Les commentaires XML ne sont pas utilisés dans la présente spécification.

Noter que sauf si CanonicalizationMethod retire les commentaires au sein de SignedInfo ou tout autre XML référencé (ce que fait la [RFC3076]) ils seront signés. Par conséquent, si ils sont conservés, un changement aux commentaires va causer un échec de la signature. De même, la signature XML sur toutes données XML sera sensible aux changements de commentaires, sauf si est spécifiée une méthode de canonisation/transformation qui ignore les commentaires, telle que le XML canonique [RFC3076].

6. Algorithmes

La présente section identifie les algorithmes utilisés avec la spécification de signature numérique XML. Les entrées contiennent l'identifiant à utiliser dans les éléments Signature, une référence à la spécification formelle, et les définitions, lorsque il en est d'applicables, pour la représentation des clés et des résultats des opérations de chiffrement.

6.1 Identifiants d'algorithme et exigences de mise en œuvre

Les algorithmes sont identifiés par les URI qui apparaissent comme attribut de l'élément qui identifie le rôle des algorithmes (DigestMethod, Transform, SignatureMethod, ou CanonicalizationMethod). Tous les algorithmes utilisés ici prennent des paramètres, mais dans de nombreux cas, les paramètres sont implicites. Par exemple, une SignatureMethod reçoit implicitement deux paramètres : les informations de clés et le résultat de CanonicalizationMethod. Les paramètres supplémentaires explicites d'un algorithme apparaissent comme éléments de contenu au sein de l'élément de rôle de l'algorithme. De tels éléments de paramètres ont un nom d'élément descriptif, qui est fréquemment spécifique de l'algorithme, et DOIT être dans l'espace de noms de signature XML ou dans un espace de noms spécifique de l'algorithme.

La présente spécification définit un ensemble d'algorithmes, leurs URI, et les exigences de leur mise en œuvre. Les exigences sont spécifiées sur la mise en œuvre, non sur les exigences pour l'utilisation de la signature. De plus, le mécanisme est extensible ; des algorithmes de remplacement peuvent être utilisés par les applications de signature.

Résumé
1. SHA1 exigé
http://www.w3.org/2000/09/xmldsig#sha1

Codage
1. base64 exigé
http://www.w3.org/2000/09/xmldsig#base64

MAC
1. HMAC-SHA1 exigé
http://www.w3.org/2000/09/xmldsig#hmac-sha1

Signature
1. DSA avec SHA1 (DSS) exigé
http://www.w3.org/2000/09/xmldsig#dsa-sha1
2. Recommended RSAwithSHA1
http://www.w3.org/2000/09/xmldsig#rsa-sha1

Canonisation
1. XML canonique exigé (omet les commentaires)
http://www.w3.org/TR/2001/REC-xml-c14n-20010315
2. XML canonique avec commentaires recommandé
http://www.w3.org/TR/2001/REC-xml-c14n-20010315#WithComments

Transformation
1. XSLT facultatif
http://www.w3.org/TR/1999/REC-xslt-19991116
2. Xpath Recommandé
http://www.w3.org/TR/1999/REC-xpath-19991116
3. Signature enveloppée exigée *
http://www.w3.org/2000/09/xmldsig#enveloped-signature

* La transformation Signature enveloppée retire l'élément Signature du calcul de la signature lorsque la signature est dans le contenu qui va être signé. Cela PEUT être mis en œuvre via la spécification XPath RECOMMANDÉE spécifiée au paragraphe 6.6.4 "Transformation Signature enveloppée" ; cela DOIT avoir le même effet que celui spécifié par la transformation XPath.

6.2 Résumés de message

Un seul algorithme de résumé est défini ici. Cependant, il est prévu qu'un ou plusieurs algorithmes de résumé forts supplémentaires seront développés en connexion avec l'effort US "Advanced Encryption Standard". L'utilisation de MD5 [RFC1321] N'EST PAS RECOMMANDÉE parce que des avancées récentes en cryptanalyse ont fait naître des doutes sur sa force.

6.2.1 SHA-1

Identifiant : http://www.w3.org/2000/09/xmldsig#sha1

L'algorithme SHA-1 [SHA-1] ne prend pas de paramètre explicite. Un exemple d'élément de DigestAlg SHA-1 est :

```
<DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
```

Un résumé SHA-1 est une chaîne de 160 bits. Le contenu de l'élément DigestValue devra être le codage en base64 de cette chaîne binaire vue comme un flux d'octets de 20 octets. Par exemple, l'élément DigestValue pour le résumé de message :

```
A9993E36 4706816A BA3E2571 7850C26C 9CD0D89D
```

tiré de l'Appendice A de la norme SHA-1 serait :

```
<DigestValue>qZk+NkcGgWq6PiVxeFDCbJzQ2J0=</DigestValue>
```

6.3 Codes d'authentification de messages

Les algorithmes de MAC prennent deux paramètres implicites, leur matériel de clé déterminé à partir de KeyInfo et le flux d'octets produit par la CanonicalizationMethod. Les MAC et les algorithmes de signature sont syntaxiquement identiques mais un MAC implique une clé secrète partagée.

6.3.1 HMAC

Identifiant : http://www.w3.org/2000/09/xmldsig#hmac-sha1

L'algorithme HMAC [RFC2104] prend comme paramètre la longueur de troncature en bits ; si le paramètre n'est pas spécifié, alors tous les bits du hachage sont le résultat. Un exemple d'un élément SignatureMethod de HMAC est :

```
<SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#hmac-sha1">
      <HMACOutputLength>128</HMACOutputLength>
  </SignatureMethod>
```

Le résultat de l'algorithme HMAC est en final le résultat (éventuellement tronqué) de l'algorithme de résumé choisi. Cette valeur devra être codée en base64 de la même façon directe que le résultat de l'algorithme de résumé. Exemple : l'élément SignatureValue pour le résumé HMAC-SHA1

```
9294727A 3638BB1C 13F48EF8 158BFC9D (tiré des vecteurs d'essai de la [RFC2104]) serait :

<SignatureValue>kpRyejY4uxwT9I74FYv8nQ==</SignatureValue>
```

Définition du schéma :

```
<simpleType name="HMACOutputLengthType">
  <restriction base=«entier»/>
</simpleType>
```

DTD : `<!ELEMENT HMACOutputLength (#PCDATA)>`

6.4 Algorithmes de signature

Les algorithmes de signature prennent deux paramètres implicites, leur matériel de clé déterminé à partir de KeyInfo et flux d'octets produit par CanonicalizationMethod. Les algorithmes de signature et de MAC sont syntaxiquement identiques mais la signature implique un chiffrement à clé publique.

6.4.1 DSA

Identifiant : http://www.w3.org/2000/09/xmldsig#dsa-sha1

L'algorithme DSA [DSS] ne prend pas de paramètre explicite. Un exemple d'élément SignatureMethod DSA est :

```
<SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#dsa-sha1"/>
```

Le résultat de l'algorithme DSA consiste en une paire d'entiers qu'on appelle usuellement la paire (r, s). La valeur de la signature consiste en le codage base64 de l'enchaînement de flux de deux octets qui résultent respectivement du codage en octet des valeurs r et s dans cet ordre. La conversion d'entier en flux d'octets doit être faite conformément à l'opération I2OSP définie dans la [RFC2437] avec un paramètre l égal à 20. Par exemple, l'élément SignatureValue pour une signature DSA (r, s) avec les valeurs spécifiées en hexadécimal :

r = 8BAC1AB6 6410435C B7181F95 B16AB97C 92B341C0
s = 41E2345F 1F56DF24 58F426D1 55B4BA2D B6DCD8C8

d'après l'exemple de l'Appendice 5 de la norme DSS serait

<SignatureValue> i6watmQQQ1y3GB+VsWq5fJKzQcBB4jRfH1bfJFj0JtFVtLotttzYyA== </SignatureValue>

6.4.2 PKCS1 (RSA-SHA1)

Identifiant : http://www.w3.org/2000/09/xmldsig#rsa-sha1

L'expression "algorithme RSA" telle qu'utilisée dans le présent document se réfère à l'algorithme RSASSA-PKCS1-v1_5 décrit dans la [RFC2437]. L'algorithme RSA ne prend pas de paramètre explicite. Un exemple d'un élément RSA SignatureMethod est :

<SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>

Le contenu SignatureValue pour une signature RSA est le codage en base64 [RFC2045] de la chaîne d'octets calculée selon le paragraphe 8.1.1 de la [RFC24371] "Génération de signature pour le schéma de signature RSASSA-PKCS1-v1_5". Comme spécifié dans la fonction EMSA-PKCS1-V1_5-ENCODE du paragraphe 9.2.1 de la [RFC2437], la valeur entrée dans la fonction de signature DOIT contenir en en-tête un identifiant d'objet algorithme pour la fonction de hachage, mais la disponibilité d'un analyseur ASN.1 et la reconnaissance des OID ne sont pas exigées d'un vérificateur de signature. La représentation de PKCS#1 v1.5 apparaît sous la forme :

CRYPT (PAD (ASN.1 (OID, DIGEST (données))))

Noter que l'ASN.1 bourré sera de la forme suivante :

01 | FF* | 00 | préfixe | hachage

où "|" figure l'enchaînement, "01", "FF", et "00" sont des octets fixés de la valeur hexadécimale correspondante, "hachage" est le résumé SHA1 des données, et "préfixe" est le préfixe de désignation d'algorithme SHAI BER ASN.1 exigé par PKCS1 [RFC2437], c'est-à-dire,

hex 30 21 30 09 06 05 2B 0E 03 02 1A 05 00 04 14

Ce préfixe est inclus pour rendre plus facile l'utilisation de bibliothèques standard de cryptographie. L'octet FF DOIT être répété le nombre maximum de fois pour que la valeur de la quantité qui va être chiffrée soit d'un octet plus courte que le modulo RSA.

La chaîne base64 [RFC2045] résultante est la valeur du nœud texte fils de l'élément SignatureValue, par exemple,

```
<SignatureValue>
 IWijxQjUrcXBYoCei4QxjWo9Kg8D3p9tlWoT4t0/gyTE96639
 In0FZFY2/rvP+/bMJ01EArmKZsR5VW3rwoPxw=
</SignatureValue>
```

6.5 Algorithmes de canonisation

Si la canonisation est effectuée sur les octets, les algorithmes de canonisation prennent deux paramètres implicites : le contenu et son jeu de caractères. Le jeu de caractères est déduit conformément aux règles des protocoles de transport et des types de support (par exemple, la [RFC2376] définit les types de support pour XML). Ces informations sont nécessaires pour signer et vérifier correctement les documents et exige souvent une configuration soignée du côté serveur.

Divers algorithmes de canonisation exigent une conversion à la [RFC3629]. Les deux algorithmes ci-dessous comprennent au moins les [RFC3629] et [RFC2781] comme codages d'entrée. On RECOMMANDE que les algorithmes extérieurs spécifiés fassent la même chose. La connaissance d'autres codages est FACULTATIVE.

Divers algorithmes de canonisation transcodent d'un codage non Unicode en Unicode. Les deux algorithmes ci-dessous effectuent la normalisation de texte durant le transcodage [NFC]. On RECOMMANDE que les algorithmes de canonisation externes spécifiés fassent de même. (Noter qu'il peut y avoir des ambiguïtés dans les conversions de jeux de caractères existants en Unicode, par exemple, voir le profil XML japonais [XML-Jap].)

6.5.1 XML canonique

Identifiant pour XML canonique EXIGÉ (omet les commentaires) : http://www.w3.org/TR/2001/REC-xml-c14n-20010315

Identifiant pour XML canonique avec commentaires :
http://www.w3.org/TR/2001/REC-xml-c14n-20010315#WithComments

Un exemple d'élement de canonisation XML est :

```
<CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
```

La spécification normative du XML canonique est la [RFC3076]. L'algorithme est capable de prendre en entrée un flux d'octets ou un ensemble-nœud XPath (ou une solution de remplacement suffisamment fonctionnelle). L'algorithme produit un flux d'octets en sortie. Le XML canonique est facilement paramétrable (via un URI additionnel) pour omettre ou conserver les commentaires.

6.6 Algorithmes de transformation

Un algorithme Transform a un seul paramètre implicite : un flux d'octets provenant de Reference ou du résultat d'une transformation antérieure.

Il est fortement recommandé aux développeurs d'applications de prendre en charge toutes les transformations énumérées dans ce paragraphe comme RECOMMANDÉ sauf si l'environnement de l'application a des contraintes de ressources qui rendraient impraticable une telle prise en charge. La conformité à cette recommandation va maximiser l'interopérabilité des applications et des bibliothèques devraient être disponibles pour permettre la prise en charge de ces transformations dans les applications sans grands développements.

6.6.1 Canonisation

Tout algorithme de canonisation qui peut être utilisé pour CanonicalizationMethod (comme ceux qui sont au paragraphe 6.5 "Algorithmes de canonisation") peuvent être utilisés comme transformation.

6.6.2 Base64

Identifiant : http://www.w3.org/2000/09/xmldsig#base64

La spécification normative pour les transformations de décodage base64 est la [RFC2045]. L'élément Transform base64 n'a pas de contenu. L'entrée est décodée par les algorithmes. Cette transformation est utile si une application a besoin de signer les données brutes associées au contenu codé d'un élément.

Cette transformation exige un flux d'octets en entrée. Si un ensemble-nœud XPath (ou une solution de remplacement suffisamment fonctionnelle) est donnée en entrée, il est alors converti en un flux d'octets en effectuant des opérations logiquement équivalentes à 1) appliquer une transformation XPath avec l'expression self::text(), puis 2) prendre la valeur de chaîne de l'ensemble-nœud. Donc, si un élément XML est identifié par un simple nom XPointer dans l'URI de référence, et si son contenu consiste seulement en données de caractères codés en base64, cette transformation ôte alors automatiquement les étiquettes de début et de fin de l'élément identifié et de tous éléments descendants ainsi que de tout commentaire descendant et toutes instructions de traitement. Le résultat de cette transformation est un flux d'octets.

6.6.3 Fitrage de XPath

Identifiant : http://www.w3.org/TR/1999/REC-xpath-19991116

La spécification normative pour une évaluation d'expression XPath est [XPath]. L'expression XPath à évaluer apparaît comme le contenu en caractères d'un élément enfant de paramètre de transformation nommé XPath.

L'entrée requise par cette transformation est un ensemble-nœud XPath. Noter que si l'entrée réelle est un ensemble-nœud XPath résultant d'un URI nul ou de déréférence de nom nu XPointer, les nœuds de commentaires auront alors été omis. Si l'entrée réelle est un flux d'octets, l'application DOIT alors convertir le flux d'octets en un ensemble-nœud XPath convenable pour être utilisé par du XML canonique avec commentaires. (Une application suivante de l'algorithme XML canonique EXIGÉ ôterait ces commentaires.) En d'autres termes, l'ensemble-nœud d'entrée devrait être équivalent à celui qui serait créé par le processus suivant :

1. Initialiser un contexte d'évaluation XPath en réglant le nœud initial égal au nœud racine du document XML d'entrée, et régler la position et la taille du contexte à 1.
2. Évaluer l'expression XPath (//. | //@* | //namespace::*)
   L'évaluation de cette expression inclut tous les nœuds du document (y compris les commentaires) dans l'ensemble-nœud représentant le flux d'octets.

Le résultat de la transformation est aussi un ensemble-nœud XPath. L'expression Xpath qui apparaît dans le paramètre XPath est évaluée une fois pour chaque nœud dans l'ensemble-nœud d'entrée. Le résultat est converti en un booléen. Si le booléen est vrai, le nœud est alors inclus dans l'ensemble-nœud de sortie. Si le booléen est faux, le nœud est alors omis de l'ensemble-nœud de sortie.

Note: Même si l'ensemble-nœud de sortie avait eu des commentaires retirés, les nœuds de commentaire existent toujours dans l'arborescence d'analyse sous-jacente et peuvent séparer les nœuds de texte. Par exemple, la balise <e>Hello, <!-- comment -->world!</e> contient deux nœuds de texte. Donc, l'expression self::text()[string()="Hello, world!"] va echouer. Si ce problème devait survenir dans l'application, il pourrait être résolu soit par la canonisation du document avant la transformation XPath pour retirer physiquement les commentaires, soit en confrontant le nœud à la valeur de chaîne de l'élément parent (par exemple, en utilisant l'expression self::text()[string(parent::e)="Hello, world!"]).

Le principal objet de cette transformation est de s'assurer que seuls les changements définis spécifiquement pour le document XML d'entrée sont permis après l'apposition de la signature. Cela est fait en omettant précisément les nœuds auxquels il est permis de changer une fois que la signature est apposée, et en incluant tous les autres nœuds d'entrée dans le résultat. Il est de la responsabilité de l'auteur de l'expression XPath d'inclure tous les nœuds dont le changement pourrait affecter l'interprétation du résultat de la transformation dans le contexte d'application.

Un scénario important serait celui d'un document exigeant deux signatures enveloppées. Chaque signature doit s'omettre elle-même de ses propres calculs de résumé, mais il est aussi nécessaire d'exclure le second élément de signature des calculs de résumé de la première signature afin que l'ajout de la seconde signature ne casse pas la première signature.

La transformation XPath établit le contexte d'évaluation suivant pour chaque nœud de l'ensemble-nœud d'entrée :

* Un nœud de contexte égal à un nœud de l'ensemble-nœud d'entrée.
* Une position de contexte initialisée à 1.
* Une taille de contexte initialisée à 1.
* Une bibliothèque de fonctions égale à l'ensemble de fonctions défini dans [XPath] plus une fonction désignée ici.
* Un ensemble de liens variable. Aucun moyen de les initialiser n'est défini. Donc, l'ensemble de liens variable utilisé pour l'évaluation de l'expression XPath est vide, et l'utilisation d'une référence variable dans l'expression XPath résulte en une erreur.
* L'ensemble des déclarations d'expace de noms en portée pour l'expression XPath.

Il résulte du réglage du nœud de contexte que l'expression XPath qui apparaît dans cette transformation sera assez similaire à celles utilisées dans [XSLT], sauf que la taille et la position sont toujours 1 pour refléter le fait que la transformation va automatiquement visiter chaque nœud (dans XSLT, on invoque de façon récurrente la commande apply-templates pour visiter les nœud de l'arborescence d'entrées).

La fonction here() est définie comme suit :

Fonction : ensemble-nœud here()

La fonction here retourne un ensemble-nœud contenant l'attribut ou le nœud d'instructions de traitement ou l'élément parent du nœud texte qui porte directement l'expression XPath. Cette expression résulte en une erreur si l'expression XPath contenante n'apparaît pas dans le même document XML par rapport auquel l'expression XPath est évaluée.

Par exemple, considérons la création d'une signature enveloppée (un élément Signature qui est un descendant d'un élément à signer). Bien que le contenu signé n'ait pas été changé après la signature, les éléments au sein de l'élément Signature

changent (par exemple, la valeur de résumé doit être mise à l'intérieur de DigestValue et la SignatureValue doit être ensuite calculée). Un moyen d'empêcher ces changements d'invalider la valeur de résumé dans DigestValue est d'ajouter une transformation XPath qui omet tous les éléments Signature et leurs descendants.
Par exemple,

```
<Document>
...
<Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
  <SignedInfo>
    ...
    <Reference URI="">
      <Transforms>
        <Transform
Algorithm="http://www.w3.org/TR/1999/REC-xpath-19991116">
          <XPath xmlns:dsig="&dsig;">
          not(ancestor-or-self::dsig:Signature)
          </XPath>
        </Transform>
      </Transforms>
      <DigestMethod
Algorithme="http://www.w3.org/2000/09/xmldsig#sha1"/>
      <DigestValue></DigestValue>
    </Reference>
  </SignedInfo>
  <SignatureValue></SignatureValue>
</Signature>
...
</Document>
```

Du fait de l'URI Reference nul dans cet exemple, l'ensemble-nœud d'entrée de la transformation XPath contient tous les nœuds dans l'arborescence d'analyse entière commençant au nœud racine (excepté les nœuds commentaires). Pour chaque nœud dans cet ensemble-nœud, le nœud est inclus dans l'ensemble-nœud de sortie sauf si le nœud ou un de ses ancêtres, a une étiquette de Signature qui est dans l'espace de nom donné par le texte de remplacement pour l'entité &dsig;.

Une solution plus élégante utilise la fonction here pour n'omettre que la Signature contenant la transformation XPath, permettant ainsi aux signatures enveloppées de signer d'autres signatures. Dans l'exemple ci-dessus, en utilisant l'élément XPath :

```
<XPath xmlns:dsig="&dsig;">
count(ancestor-or-self::dsig:Signature | here()/ancestor::dsig:Signature[1]) >
count(ancestor-or-self::dsig:Signature)</XPath>
```

Comme l'opérateur d'égalité XPath convertit les ensembles nœuds en valeurs de chaîne avant la comparaison, on doit plutôt utiliser l'opérateur XPath union (|). Pour chaque nœud du document, l'expression du prédicat est vraie si et seulement si l'ensemble-nœud contenant le nœud et ses éléments Signature ancêtres n'inclut pas l'élément Signature enveloppé contenant l'expression XPath (l'union ne produit pas un ensemble plus grand si l'élément Signature enveloppé est dans l'ensemble-nœud donné par ancestor-or-self::Signature).

6.6.4 Transformation de signature enveloppée

Identifiant : http://www.w3.org/2000/09/xmldsig#enveloped-signature

Une transformation de signature enveloppée T retire la totalité de l'élément Signature contenant T du calcul de résumé de l'élément de référence qui contient T. La totalité de la chaîne de caractères utilisée par un processeur XML pour confronter la signature avec l'élément de production XML est retirée. Le résultat de la transformation est équivalent au résultat qui aurait résulté du remplacement de T par une transformation XPath contenant l'élément paramètre XPath suivant :

```
<XPath xmlns:dsig="&dsig;">
count(ancestor-or-self::dsig:Signature | here()/ancestor::dsig:Signature[1]) >
count(ancestor-or-self::dsig:Signature)</XPath>
```

Les exigences d'entrée et de sortie de cette transformation sont identiques à celles de la transformation XPath, mais ne peut

être appliquée qu'à un ensemble-nœud provenant de son document XML parent. Noter qu'il n'est pas nécessaire d'utiliser un évaluateur d'expression XPath pour créer cette transformation. Cependant, cette transformation DOIT produire son résultat exactement de la même manière que la transformation XPath paramétrée par l'expression XPath ci-dessus.

6.6.5 Transformation XSLT

Identifiant : http://www.w3.org/TR/1999/REC-xslt-19991116

La spécification normative pour les transformations XSL est [XSLT]. La spécification d'un élément de feuille de style qualifiée par un espace de noms, qui DOIT être le seul enfant de l'élément Transform, indique que la feuille de style spécifiée devrait être utilisée. Le modèle de traitement XSLT détermine si cela provoque le traitement en ligne d'une déclaration XSLT locale au sein de la ressource ; l'application ordonnée de plusieurs feuilles de style peut exiger plusieurs transformations. Aucune disposition particulière n'est prise pour l'identification d'une feuille de style distante à un certain URI parce que il peut être communiqué via un xsl:include ou un xsl:import au sein de la feuille de style fille de la transformation.

Cette transformation exige un flux d'octets en entrée. Si l'entrée réelle est un ensemble-nœud XPath, l'application de signature devrait alors essayer de le convertir en octets (en appliquant XML canonique) comme décrit dans le modèle de traitement de référence (paragraphe 4.3.3.2).

Le résultat de cette transformation est un flux d'octets. Les règles de traitement de la feuille de style XSL ou de l'élément Transform sont établies dans la spécification [XSLT]. On RECOMMANDE que les auteurs de transformations XSLT utilisent une méthode de sortie de xml pour XML et HTML. Comme les mises en œuvre de XSLT ne produisent pas de mises en série cohérentes de leurs résultats, on RECOMMANDE de plus d'insérer une transformation après la transformation XSLT pour canoniser le résultat. Ces étapes vont aider à assurer l'interopérabilité des signatures résultantes entre les applications qui prennent en charge la transformation XSLT. Noter que si le résultat est en fait HTML, le résultat de ces étapes sera logiquement équivalent [XHTML].

7. Canonisation XML et considérations sur les contraintes de syntaxe

Les signatures numériques ne fonctionnent que si les calculs de vérification sont effectués exactement sur les mêmes bits que ceux des calculs de signature. Si la représentation de la surface des données signées peut changer entre la signature et la vérification, on doit utiliser des moyens de normalisation des aspects changeables avant la signature et la vérification. Par exemple, même pour du simple texte ASCII, il y a au moins trois séquences largement utilisées de terminaison de ligne. Si il est possible que du texte signé soit modifié d'une convention de terminaison de ligne à une autre entre le moment de la signature et celui de la vérification de la signature, les terminaisojs de ligne ont besoin d'être canonisées en forme standard avant de signer et de vérifier, sinon les signatures vont échouer.

XML est soumis à des changements de représentation de surface et à un traitement qui va éliminer certaines informations de surface. Pour cette raison, les signatures numériques XML ont une disposition pour indiquer les méthodes de canonisation de la signature afin qu'un vérificateur puisse utiliser la même canonisation que le signataire.

Dans la présente spécification, on distingue la canonisation d'un élément Signature et celle des autres objets de données XML signés. Il est possible qu'un document XML isolé soit traité comme si il y avait des données binaires afin qu'aucun changement ne puisse survenir. Dans ce cas, le résumé du document ne va pas changer et il n'a pas besoin d'être canonisé si il est signé et vérifié comme tel. Cependant, le XML qui est lu et traité en utilisant les techniques standard d'analyse et de traitement XML est fréquemment changé de telle sorte que certaines de ses informations de représentation de surface sont perdues ou modifiées. En particulier, cela va se produire dans de nombreux cas pour les éléments Signature et SignedInfo inclus car ils, et éventuellement le document XML qui les entoure, seront traités comme du XML.

De même, ces considérations s'appliquent aux éléments Manifest, Object, et SignatureProperties si ces éléments ont été résumés, leur DigestValue est à vérifier, et ils seront traités comme du XML.

Le type de changements dans XML qui peut devoir être canonisé peut se diviser en quatre catégories. Il y a ceux qui se rapportent au [XML] de base, tel que décrit au paragraphe 7.1 ci-dessous. Il y a ceux qui se rapportent aux traitements [DOM], [SAX], ou similaires tels que décrits au paragraphe 7.2. Troisièmement, il y a la possibilité de conversion de jeu de caractères codés, comme entre UTF-8 et UTF-16, pour lesquels des processeurs conformes à [XML] sont exigés pour prendre en charge ce qui est décrit au paragraphe suivant. Et quatrièmement, il y a des changements qui se rapportent au contexte de déclaration d'espace de noms et d'attribut d'expace de noms XML comme décrit au paragraphe 7.3.

Tout algorithme de canonisation devrait donner un résultat dans un jeu de caractères codé spécifique fixé. Tous les

algorithmes de canonisation identifiés dans le présent document utilisent UTF-8 (sans marque d'ordre des octets (BOM, *byte order mark*)) et ne fournissent pas la normalisation de caractères. On RECOMMANDE que les applications de signature créent un contenu XML (éléments Signature et leurs descendants/contenu) en forme de normalisation C [NFC] et vérifient que tout XML consommé est aussi sous cette forme ; (sinon, les signatures peuvent ensuite échouer à la validation). De plus, aucun de ces algorithmes ne fournit la normalisation du type de données. Les applications qui normalisent les types de données dans divers formats (par exemple, (vrai, faux) ou (1,0)) peuvent n'être pas capables de valider les signatures de chacune des autres.

7.1 XML 1.0, contraintes de syntaxe et canonisation

XML 1.0 [XML] définit une interface où une application conforme qui lit du XML reçoit certaines informations de la part de cet XML et pas certaines autres. En particulier,

1. les terminaisons de ligne sont normalisées au seul caractère #xA en abandonnant le caractère #xD si ils sont immédiatement suivis par un #xA et en les remplaçant par #xA dans tous les autres cas,
2. les attributs manquants déclarés avoir des valeurs par défaut sont fournis à l'application comme si ils étaient présents avec la valeur par défaut,
3. les références à des caractères sont remplacées par le caractère correspondant,
4. les références à des entités sont remplacées par l'entité déclarée correspondante,
5. les valeurs d'attribut sont normalisées par
   5.1 le remplacement des références de caractère et d'entité comme indiqué ci-dessus,
   5.2 le remplacement des occurrences de #x9, #xA, et #xD par #x20 (espace) excepté la séquence #xD#xA qui est remplacée par une seule espace, et
   5.3 si l'attribut n'est pas déclaré comme CDATA, le retrait de toutes les espaces en tête et en queue et le remplacement de toutes les suites intérieurrs d'espaces par une seule espace.

Noter que les items (2), (4), et (5.3) dépendent de la présence d'un schéma, DTD ou déclarations similaires. Le type d'élément Signature est valide selon le schéma relâché [XML-sch], par conséquent, le XML externe ou même le XML au sein du même document que la signature peut être (seulement) bien formé ou d'un autre espace de noms (lorsque c'est permis par le schéma de signature) les éléments notés peuvent n'être pas présents. Donc, une signature avec un tel contenu ne sera vérifiable par d'autres applications de signature que si les contraintes syntaxiques suivantes sont observées lors de la génération de tout matériel signé incluant l'élement SignedInfo :

1. les attributs qui ont des valeurs par défaut sont explicitement présents ,
2. toutes les références d'entité (excepté "amp", "lt", "gt", "apos", "quot", et autres entités de caractères non représentables dans le codage choisi) sont développées,
3. la valeur d'attribut espace blanche est normalisée.

7.2 Traitement et canonisation de DOM/SAX

En plus des contraintes de canonisation et de syntaxe exposées ci-dessus, de nombreuses applications XML utilisent le modèle d'objet Document [DOM] ou l'API simple pour XML [SAX]. DOM transpose XML en une structure arborescente de nœuds et suppose normalement qu'elle sera utilisée sur la totalité d'un document avec un traitement ultérieur effectué sur cette arborescence. SAX convertit XML en une série d'événements tels qu'une étiquette de début, un contenu, etc. Dans l'un et l'autre cas, de nombreuses caractéristiques de surface telles que l'ordre des attributs et les espaces blanches non significatives au sein des étiquettes de début/fin sont perdues. De plus, les déclarations d'espace de noms sont transposées sur les nœuds auxquels elles s'appliquent, perdant les préfixes d'espace de nom du texte source, dans la plupart des cas, perdant l'information sur où apparaissaient les déclarations d'espace de noms dans l'instance originale.

Si une signature XML doit être produite ou vérifiée dans un système utilisant un traitement DOM ou SAX, une méthode canonique est nécessaire pour mettre à la suite les parties pertinentes d'une arborescence DOM ou une séquence d'événements SAX. Les spécifications de canonisation XML, telles que la [RFC3076], ne se fondent que sur des informations qui sont préservées par DOM et SAX. Pour qu'une signature XML soit vérifiable par une mise en œuvre qui utilise DOM ou SAX, non seulement les contraintes syntaxiques XML 1.0 données dans les paragraphes qui précèdent doivent être suivies, mais une canonisation XML appropriée DOIT être spécifiée afin que le vérificateur puisse remettre à la suite les entrées traitées pas DOM/SAX dans le même flux d'octets que celui où elles ont été signées.

7.3 Contexte d'espace de noms et signatures portables

Dans [XPath] et par conséquent dans le modèle de données XML canoniques, un élément a des nœuds d'espace de noms

qui correspondent aux déclarations qui sont dans l'élément et ses ancêtres :

"Note :Un élément E a des nœuds d'espace de noms qui représentent ses déclarations d'espace de noms aussi bien que toutes les déclarations d'espace de noms faites par ses ancêtres qui n'ont pas été écrasées dans les déclarations de E, l'espace de noms par défaut si il n'est pas vide, et la déclaration du préfixe xml." [RFC3076]

Lorsque on met en série un élément Signature ou des données XML signées qui sont les enfants d'autres éléments qui utilisent ces modèles de données, cet élément Signature et ses enfants peuvent contenir des déclarations d'espace de noms provenant de son contexte ancêtre. De plus, les algorithmes XML canonique et XML canonique avec commentaires importent tous les attributs espace de noms xml (tels que xml:lang) du plus proche ancêtre dans lequel ils sont déclarés au nœud apex de XML canonisé sauf si ils sont déjà déclarés sur ce nœud. Cela peut déjouer l'intention du signataire de créer une signature dans un contexte qui reste valide dans un autre. Par exemple, soit une signature qui serait un fils de B et un petit fils de A :

```
<A xmlns:n1="&foo;">
  <B xmlns:n2="&bar;">
    <Signature xmlns="&dsig;">   ...
      <Reference URI="#signme"/> ...
    </Signature>
    <C ID="signme" xmlns="&baz;"/>
  </B>
</A>
```

Lorsque l'élément B ou l'élément signé C est déplacé dans une enveloppe [SOAP] pour son transport :

```
  <SOAP:Envelope
xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope/">
    ...
    <SOAP:Body>
      <B xmlns:n2="&bar;">
        <Signature xmlns="&dsig;">
          ...
        </Signature>
        <C ID="signme" xmlns="&baz;"/>
      </B>
    </SOAP:Body>
  </SOAP:Envelope>
```

La forme canonique de la signature dans ce contexte va contenir de nouvelles déclarations d'espace de noms provenant du contexte de l'enveloppe SOAP, rendant la signature invalide. Aussi, la forme canonique va manquer des déclarations d'espace de noms qu'elle aurait pû avoir à l'origine du contexte de l'élément A, invalidant aussi la signature. Pour éviter ces problèmes, l'application peut :

1. s'appuyer sur l'application enveloppante pour séparer correctement le corps (la charge utilie de signature) du contexte (l'enveloppe) avant que la signature soit validée.
2. Utiliser une méthode de canonisation qui "repousse/exclu" au lieu de "attirer" le contexte ancêtre. La [RFC3076] attire à dessein un tel contexte.

8. Considérations pour la sécurité

La spécification de signature XML fournit un mécanisme de signature numérique très souple. Les mises en œuvre doivent prêter attention aux modèles de menace de leur application et aux facteurs qui suivent.

8.1 Transformations

Une exigence de la présente spécification est de permettre que les signatures "s'appliquent à une partie ou à la totalité d'un document XML". (Voir le paragraphe 3.1.3 de la [RFC2807].) Le mécanisme de transformation satisfait à cette exigence en permettant de signer les données déduites du traitement du contenu de la ressource identifiée. Par exemple, les applications qui souhaitent signer un formulaire, mais permettent aux utilisateurs d'entrer des données dans un champ limité sans invalider une signature antérieure sur le formulaire peuvent utiliser [XPath] pour exclure ces portions que l'utilisateur doit changer. Les transformations peuvent être arbitrairement spécifiées et peuvent inclure des transformations de codage, des

instructions de canonisation ou même des transformations XSLT. Trois recommandations sont formulées par rapport à cette caractéristique dans les paragraphes qui suivent.

Noter que le comportement de validation de cœur ne confirme pas que les données signées aient été obtenues en appliquant chaque étape des transformations indiquées. (Bien qu'il vérifie que le résumé du contenu résultant correspond bien à celui spécifié dans la signature.) Par exemple, certaines applications peuvent se satisfaire de la vérification d'une signature XML sur une copie en antémémoire de données déjà transformées. D'autres applications pourront exiger que le contenu soit fraîchement déréférencé et transformé.

8.1.1 Seul ce qui est signé est sûr

Tout d'abord, il est évident que les signatures sur un document transformé ne sécurisent aucune des informations éliminées par les transformations : seul ce qui est signé est sûr.

Noter que l'utilisation du XML canonique [RFC3076] assure que toutes les entités internes et les espaces de noms XML sont développés au sein du contenu qui est signé. Toutes les entités sont remplacées par leur définition et la forme canonique représente explicitement l'espace de noms dont aurait autrement hérité un élément. Les applications qui ne canonisent pas le contenu XML (en particulier l'élément SignedInfo) NE DEVRAIENT PAS utiliser d'entités internes et DEVRAIENT représenter explicitemnt l'espace de noms au sein du contenu signé car elles ne peuvent pas compter que la canonisation va le faire pour elles. Aussi, les usagers qui se soucient de l'intégrité des définitions de type d'élément associées à l'instance XML à signer peuvent souhaiter signer aussi ces définitions (c'est-à-dire, le schéma, le DTD, ou la description en langage naturel associée à l'espace de noms/identifiant).

Ensuite, une enveloppe contenant des informations signées n'est pas sécurisée par la signature. Par exemple, lorsque une enveloppe chiffrée contient une signature, celle-ci ne protège pas l'authenticité ou l'intégrité des en-têtes d'enveloppe non signés ni sa forme chiffrée, elle ne sécurise que le texte en clair effectivement signé.

8.1.2 Seul ce qui est "vu" devrait être signé

De plus, la signature sécurise toutes les informations introduites par la transformation : seul ce qui est "vu" (c'est ce qui est représenté à l'utilisateur via un support visuel, auditif ou autre) devrait être signé. Si la signature est destinée à véhiculer le jugement ou le consentement d'un utilisateur (un mécanisme automatique ou une personne) il est alors normalement nécessaire de sécuriser aussi exactement que possible en pratique les informations qui ont été présentées à cet usager. Noter que ceci peut être accompli en signant littéralement ce qui a été présenté, comme les images d'écran montrées à un usager. Cependant, il peut en résulter des données qui seront difficiles à manipuler pour les logiciels ultérieurs. On peut plutôt signer les données conjointement avec tous les filtres, feuilles de style, profil de client ou autres informations qui affectent leur présentation.

8.1.3 "Voir" ce qui est signé

Tout comme un usager ne devrait signer que ce qu'il "voit", les personnes et mécanismes automatisés qui font confiance à la validité d'un document transformé sur la base d'une signature valide devraient opérer sur les données qui ont été transformées (y compris la canonisation) et signées, et non sur les données originales avant la transformation. Cette recommandation s'applique aux transformations spécifiées au sein de la signature aussi bien qu'à celles qui sont incluses au titre du document lui-même. Par exemple, si un document XML comporte une feuille de style incorporée [XSLT] c'est le document transformé qui devrait être représenté à l'usager et signé. Pour suivre cette recommandation lorsque un document fait référence à une feuille de style externe, le contenu de cette ressource externe devrait aussi être signé via une signature Reference, autrement, le contenu de cette ressource externe pourrait changer, ce qui altèrerait le document résultant sans invalider la signature.

Certaines applications peuvent fonctionner sur les données d'origine ou sur les données intermédiaires mais elles devraient faire extrêmement attention aux faiblesses potentielles introduites entre les données d'origine et les données transformées. C'est une décision de confiance sur le caractère et la signification des transformationss qu'une application a besoin de prendre avec des précautions. Considérons un algorithme de canonisation qui normalise la casse des caractères (minuscules en majuscules) ou la composition des caractères ('e et accent' en 'e accentué'). Un adversaire pourrait introduire des changements qui sont normalisés et par conséquent sans conséquence pour la validité de la signature mais matérielles pour un processeur DOM. Par exemple, en changeant la casse d'un caractère on peut influencer le résultat du choix d'un XPath. Un risque sérieux est introduit si ce changement est normalisé pour la validation de la signature mais si le processeur fonctionne sur les données originales et retourne un résultat différent de ce qui était prévu.

Il en résulte que
* tous les documents sur lesquels travaille et que génère une application de signature DOIT être en [NFC] (autrement des processeurs intermédiaires pourrait par inadvertance casser la signature)
* les normalisations de codage NE DEVRAIENT PAS être faites au titre d'une transformation de signature, ou (pour le dire autrement) si la normalisation intervient bien, l'application DEVRAIT toijours "voir" (travailler sur) la forme normalisée.

8.2 Vérifier le modèle de sécurité

La présente spécification utilise les signatures de clé publique et les codes d'authentification à hachage de clés. Ceux-ci ont des modèles de sécurité notablement différents. De plus, elle permet des algorithmes spécifiés par l'utilisateur qui peuvent avoir d'autres modèles.

Avec les signatures de clé publique, un nombre quelconque de parties peut détenir la clé publique et vérifier les signatures alors que seules les parties qui ont la clé privée peuvent créer les signatures. Le nombre de détenteurs de la clé privée devrait être minimisé et de préférence à un seul. La confiance des vérificateurs dans la clé publique qu'ils utilisent et sa liaison à l'entité ou aux capacités représentées par la clé privée correspondante est une question importante, normalement réglée par des systèmes de certificats ou d'autorité en ligne.

Les codes d'authentification à hachage de clés, fondés sur des clés secrètes, sont normalement beaucoup plus efficaces en termes d'effort de calcul exigé mais ont pour caractéristique que tous les vérificateurs doivent être en possession de la même clé que le signataire. Donc tout vérificateur peut falsifier les signatures.

La présente spécification permet les algorithmes de signature fournis par l'utilisateur et les concepteurs d'informations de matériel de clés. De tels algorithme fournis par l'utilisateur peuvent avoir des modèles de sécurité différents. Par exemple, des méthodes impliquant des éléments de biométrie qui dépendent normalement de caractéristiques physiques de l'utilisateur autorisé qui ne peuvent pas être changées comme peuvent l'être des clés publiques ou secrètes et peuvent avoir des autres différences de modèle de sécurité.

8.3 Algorithme, longueurs de clés, certificats, etc.

La force d'une signature particulière dépend de toutes les liaisons sur la chaîne de sécurité. Cela inclut les algorithmes de signature et de résumé utilisés, la force de la génération de clés [RFC1750] et de la taille de la clé, la sécurité des mécanismes d'authentification et de distribution des clés et des certificats, de la politique de validation de la chaîne de certificats, de la protection du traitement cryptographique contre l'observation et les modifications hostiles, etc.

Les applications doivent faire attention lors de l'exécution des divers algorithmes qui peuvent être spécifiés dans une signature XML et lors du traitement de tout "contenu exécutable" qui pourrait être fourni à de tels algorithmes comme paramètres, tels que les transformations XSLT. Les algorithmes spécifiés dans le présent document vont normalement être mis en œuvre via une bibliothèque de confiance, mais même là, des paramètres pervers peuvent causer des traitements ou une demande de consommation de mémoire inacceptables. Une attention encore plus grande peut être nécessaire avec les algorithmes définis par l'application.

La sécurité globale d'un système va aussi dépendre de la sécurité et de l'intégrité de ses procédures de fonctionnement, de son personnel, et de la mise en application administrative de ces procédures. Tous les facteurs énumérés dans ce paragraphe sont importants pour la sécurité globale d'un système ; cependant, la plupart sortent du domaine d'application de la présente spécification.

9. Schéma, DTD, modèle de données, et exemples valides

Instance de schéma de signature XML
http://www.w3.org/Signature/Drafts/xmldsig-core/xmldsig-core-schema.xsd
Instance de schéma XML valide fondé sur le schéma/DTD 20001024 [XML-sch].

DTD de signature XML
http://www.w3.org/Signature/Drafts/xmldsig-core/xmldsig-core-schema.dtd

Modèle de données RDF
http://www.w3.org/Signature/Drafts/xmldsig-core/xmldsig-datamodel-20000112.gif
Exemple d'objet signature XML
http://www.w3.org/Signature/Drafts/xmldsig-core/signature-exemple.xml
Exemple cryptographique XML fabriqué qui comporte un contenu étranger et valide selon le schéma, il utilise aussi schemaLocation pour aider la collecte et la validation automatisée de schéma.

Exemple de signature XML RSA
http://www.w3.org/Signature/Drafts/xmldsig-core/signature-exemple-rsa.xml
Exemple de signature XML avec des valeurs cryptographiques générées par Merlin Hughes et validées par Gregor Karlinger.

Exemple de signature XML DSA
http://www.w3.org/Signature/Drafts/xmldsig-core/signature-exemple-dsa.xml
Similaire à la précédente mais en utilisant DSA.

10. Définitions

Authentification, code (somme de contrôle protégée)
Valeur générée à partir de l'application d'une clé partagée à un message via un algorithme cryptographique qui fait qu'il a les propriétés d'authentification (et d'intégrité) du message mais pas d'authentification du signataire. Équivalent à une somme de contrôle protégée, "une somme de contrôle qui est calculée pour un objet de données par des moyens qui protègent contre les attaques actives qui tenteraient de changer la somme de contrôle pour lui faire correspondre à des changements apportés à l'objet de données." [RFC2828]

Authentification, message
Propriété, étant donné un code d'authentification/somme de contrôle protégée, qui fait que l'altération des données et/ou de la somme de contrôle, afin d'introduire des changements tout en faisant semblant de préserver l'intégrité, est toujours détectée. "Une signature devrait identifier ce qui est signé, rendant impraticable de falsifier ou d'autrement altérer la matière signée ou la signature sans détection." (Lignes directices pour les signatures numériques, [ABA].

Authentification, signataire
Propriété que l'identité du signataire est celle qu'il prétend être. "Une signature devrait indiquer qui a signé un document, message ou enregistrement, et devrait être difficile à produire par une autre personne sans autorisation." (Lignes directices pour les signatures numériques, [ABA]
Note : L'authentification du signataire est une décision d'application (par exemple, la clé de signature correspond elle réellement à une identité spécifique ?) qui est soutenue, mais sort du champ d'application de la présente spécification.

Somme de contrôle
"Valeur qui (a) est calculée par une fonction qui dépend du contenu de l'objet de données et (b) est mémorisée ou transmise avec l'objet, pour les besoins de la détection de changements dans les données." [RFC2828]

Cœur
La syntaxe et le traitement définis par la présente spécification, y compris la validation du cœur. On utilise ce terme pour distinguer d'autres sémantiques de balisage, traitement et applications, de la notre.

Objet de données (contenu/document)
Les données binaires/octet réelles sur lesquelles on travaille (transformées, résumées, ou signées) par une application – fréquemment une entité HTTP [RFC2616]. Noter que le nom propre Object désigne un élément XML spécifique. À l'occasion on se réfèrre à un objet de données comme à un contenu de document ou d'une ressource. Le terme de contenu d'élément est utilisé pour décrire les données entre le début XML et les étiquettes de fin [XML]. Le terme de document XML est utilisé pour décrire les objets de données qui se conforment à la spécification [XML].

Intégrité
"Propriété que les données n'ont pas été changées, détruites, ou perdues d'une manière non autorisée ou accidentelle." [RFC2828] Une simple somme de contrôle peut fournir l'intégrité contre des changements accidentels des données ; l'authentification de message est similaire mais protège aussi contre une attaque active pour altérer les données par laquelle un changement de la somme de contrôle est introduite de façon à correspondre au changement des données.

Objet
Élément de signature XML dans lequel des données arbitraires (non cœur) peuvent être placées. Un élément Object est simplement un type de données numériques (ou document) qui peut être signé via une Reference.

Ressource
"Une ressource peut être tout ce qui a une identité. Les exemples familiers sont un document électronique, une image, un service (par exemple, le bulletin météorologique pour "aujourd'hui" sur Paris), et une collection d'autres ressources.... La ressource est la transposition conceptuelle en une entité ou ensemble d'entités, pas nécessairement l'entité qui correspond à cette transposition à une instance particulière dans le temps. Et donc, une ressource peut rester constante même lorsque son contenu --- les entités auxquelles il correspond actuellement --- changent au fil du temps, pourvu que la transposition conceptuelle ne soit pas changée dans le processus." [RFC2396] Afin d'éviter une collision entre l'entité "terme" au sein des spécifications d'URI et de XML, nous utilisons le terme d'objet de données, de contenu ou de document pour nous référer aux bits/octets réels sur lesquels on opère.

Signature
Formellement, c'est une valeur générée à partir de l'application d'une clé privée à un message via un algorithme cryptographique tel qu'elle ait les propriétés d'intégrité, d'authentification du message et/ou d'authentification du signataire. (Cependant, on utilise parfois le terme signature de façon générique de telle sorte qu'il englobe aussi les valeurs de code d'authentification, mais nous faisons attention à bien faire la distinction lorsque la propriété d'authentification du signataire est pertinente pour l'exposé.) Une signature peut être décrite (non exclusivement) comme détachée, enveloppante, ou enveloppée.

Signature, Application
C'est une application qui met en œuvre les portions OBLIGATOIRES (EXIGE/DOIT) de la présente spécification ; ces exigences de conformité portent sur le comportement de l'application, sur la structure du type d'élément Signature et de ses enfants (y compris SignatureValue) et sur les algorithmes spécifiés.

Signature, détachée
La signature est sur du contenu externe à l'élément Signature, et peut être identifiée via un URI ou une transformation. Par conséquent, la signature est "détachée" du contenu qu'elle signe. Cette définition s'applique normalement aux objets de données séparés, mais elle inclut aussi l'instance où signature et objet de données résident au sein du même document XML mais sont des éléments frères.

Signature, enveloppante
La signature est sur du contenu qui se trouve au sein d'un élément objet de la signature elle-même. L'objet (ou son contenu) est identifié via une référence (via un identifiant ou une transformation de fragment d'URI).

Signature, enveloppée
La signature est sur le contenu XML qui contient la signature comme élément. Le contenu fournit l'élément racine de document XML. Évidemment, les signatures enveloppées doivent veiller à ne pas inclure leur propre valeur dans le calcul de SignatureValue.

Transformation
C'est le traitement des données de leur source à sa forme dérivée. Les transformations typiques sont la canonisation XML, XPath, et XSLT.

Validation, cœur
Le cœur des exigences de traitement de la présente spécification exige la validation de signature et la validation de la référence de SignedInfo.

Validation, référence
C'est la valeur hachée du contenu identifié et transformé, spécifié par Reference, qui corespond à sa DigestValue spécifiée.

Validation, signature
La SignatureValue correspond au résultat du traitement de SignedInfo avec CanonicalizationMethod et SignatureMethod comme spécifié dans Validation du cœur (paragraphe 3.2).

Validation, application de confiance
L'application détermine que la sémantique associée à une signature est valide. Par exemple, une application peut valider les horodatages ou l'intégrité de la clé du signataire – bien que ce comportement soit extérieur à ce cœur de spécification.

Appendice Changements par rapport à la RFC3075

De nombreux changements rédactionnels mineurs ont été faits. De plus les modifications substantielles suivantes sont survenues sur la base de l'expérience ou d'autres considérations :

1. Des changements mineurs mais incompatibles dans la représentation des clés DSA. En particulier, le caractère facultatif de plusieurs champs a été changé et deux champs ont été réordonnées.

2. Un changement mineur de la structure de X509Data KeyInfo pour permettre à plusieurs CRL d'être groupés avec les certificats et autres informations X509. Précédemment les CRL devaient survenirs seuls et chacun dans une structure X509Data séparée.

3. Un changement incompatible du type de PGPKeyID, qui était précédemment une chaîne, pour le type plus correct de base64Binary car il est en fait une quantité binaire.

4. Plusieurs avertissements ont été ajoutés. À noter en particulier, parce qu'il reflète un problème rencontré réellement et est le seul avertissement ajouté qui a son propre paragraphe, est l'avertissement de problèmes de canonisation lorsque le contexte d'espace de noms du matériel signé change.

Références

[1363] IEEE 1363: "Standard Specifications for Public Key Cryptography". août 2000.

[ABA] "Digital Signature Guidelines". http://www.abanet.org/scitech/ec/isc/dsgfree.html

[DOM] V. Apparao, S. Byrne, M. Champion, S. Isaacs, I. Jacobs, A. Le Hors, G. Nicol, J. Robie, R. Sutor, C. Wilson, L. Wood. "Document Object Model (DOM) Level 1 Specification". Recommandation W3C. octobre 1998. http://www.w3.org/TR/1998/REC-DOM-Level-1-19981001/

[DSS] FIPS PUB 186-2 . "Digital Signature Standard (DSS)". U.S. Department of Commerce/National Institute of Standards et Technology. http://csrc.nist.gov/publications/fips/fips186-2/fips186-2.pdf

[NFC] M. Davis, M. Drst. "TR15, Unicode Normalization Forms". Revision 18. novembre 1999. http://www.unicode.org/unicode/reports/tr15/tr15-18.html . Corrigendum à http://www.unicode.org/unicode/uni2errata/Normalization_Corrigendum.html

[RDF] D. Brickley, R.V. Guha. "Resource Description Framework (RDF) Schema Specification 1.0". Recommandation W3C candidate. mars 2000. http://www.w3.org/TR/2000/CR-rdf-schema-20000327

O. Lassila, R. Swick. "Resource Description Framework (RDF) Model et Syntax Specification". Recommandation W3C. février 1999. http://www.w3.org/TR/1999/REC-rdf-syntax-19990222/

[RFC1321] R. Rivest, "Algorithme de résumé de message MD5", avril 1992. *(Information)*

[RFC1738] T. Berners-Lee et autres, "Localisateurs uniformes de ressource (URL)", décembre 1994. *(P.S., Obsolète, voir les RFC4248 et 4266)*

[RFC1750] D. Eastlake, 3rd et autres, "Recommandations d'aléa pour la sécurité", décembre 1994. *(Info., remplacée par la RFC 4086)*

[RFC2045] N. Freed et N. Borenstein, "Extensions de messagerie Internet multi-objets (MIME) Partie 1 : Format des corps de message Internet", novembre 1996. *(D. S., MàJ par 2184, 2231, 5335.)*

[RFC2104] H. Krawczyk, M. Bellare et R. Canetti, "HMAC : Hachage de clés pour l'authentification de message", février 1997.

[RFC2119] S. Bradner, "Mots clés à utiliser dans les RFC pour indiquer les niveaux d'exigence", BCP 14, mars 1997.

[RFC2141] R. Moats, "Syntaxe des URN", mai 1997.

[RFC2253] M. Wahl, S. Kille et T. Howes, "Protocole léger d'accès à un répertoire (LDAPv3) : Représentation de chaîne UTF-8 des noms distinctifs", décembre 1997.

[RFC2376] E. Whitehead et M. Murata, "Types de support XML", juillet 1998.

[RFC2396] T. Berners-Lee, R. Fielding et L. Masinter, "Identifiants de ressource uniformes (URI) : Syntaxe générique", août 1998. (*Obsolète, voir* RFC3986)

[RFC2440] J. Callas, L. Donnerhacke, H. Finney et R. Thayer, "Format de message OpenPGP", novembre 1998. *(Obs.*

voir RFC4880)

[RFC2437] B. Kaliski et J. Staddon, "PKCS n° 1 : Spécifications de la cryptographie RSA version 2.0", octobre 1998. *(Obsolète, voir RFC3447) (Information)*

[RFC2616] R. Fielding et autres, "Protocole de transfert hypertexte -- HTTP/1.1", juin 1999. *(D.S., MàJ par 2817)*

[RFC2732] R. Hinden, B. Carpenter et L. Masinter, "Format pour les adresses littérales IPv6 dans les URL", décembre 1999.

[RFC2781] P. Hoffman et F. Yergeau, "UTF-16, un codage de la norme ISO 10646", février 2000.

[RFC2807] J. Reagle, "Exigences pour les signatures XML", juillet 2000.

[RFC2828] R. Shirey, "Glossaire de la sécurité sur l'Internet", FYI 36, mai 2000. *(Rendue obsolète par la RFC4949)*

[RFC3076] J. Boyer, "XML canonique, version 1.0", mars 2001.

[RFC3629] F. Yergeau, "UTF-8, un format de transformation de la norme ISO 10646", STD 63, novembre 2003.

[SAX] D. Megginson, et al. "SAX: The Simple API for XML". mai 1998. http://www.megginson.com/SAX/index.html (cette page est obsolète ; aller à www.saxproject.org)

[SHA-1] FIPS PUB 180-1. ""Secure Hash Standard. U.S. Department of Commerce/National Institute of Standards et Technology. http://csrc.nist.gov/publications/fips/fips180-1/fip180-1.txt

[SOAP] D. Box, D. Ehnebuske, G. Kakivaya, A. Layman, N. Mendelsohn, H. Frystyk Nielsen, S. Thatte, D. Winer. "Simple Object Access Protocol (SOAP) Version 1.1". Note W3C. mai 2001. http://www.w3.org/TR/2000/NOTE-SOAP-20000508/

[Unicode] The Unicode Consortium. "The Unicode Standard". http://www.unicode.org/unicode/standard/standard.html

[X509v3] Recommandation UIT-T X.509 version 3 (1997). "Technologies de l'information - Interconnexion des systèmes ouverts – Cadre d'authentification de l'annuaire", ISO/CEI 9594-8:1997.

[XHTML] S. Pemberton, D. Raggett, et al., "XHTML(tm) 1.0: The Extensible Hypertext Markup Language". Recommandation W3C. janvier 2000. http://www.w3.org/TR/2000/REC-xhtml1-20000126/

[XLink] S. DeRose, E. Maler, D. Orchard. "XML Linking Language". Recommandation W3C. juin 2001. http://www.w3.org/TR/2000/REC-xlink-20010627/

[XML] T. Bray, E. Maler, J. Paoli, C. M. Sperberg-McQueen, "Extensible Markup Language (XML) 1.0" (Seconde édition). Recommandation W3C. octobre 2000. http://www.w3.org/TR/2000/REC-xml-20001006

[XML-Jap] M. Murata, "XML Japanese Profile". Note W3C. avril 2000. http://www.w3.org/TR/2000/NOTE-japanese-xml-20000414/

[XML-ns] T. Bray, D. Hollander, A. Layman. "Namespaces in XML". Recommandation W3C. janvier 1999. http://www.w3.org/TR/1999/REC-xml-names-19990114

[XML-sch] D. Beech, M. Maloney, N. Mendelsohn, H. Thompson. "XML Schema Part 1: Structures". Recommandation W3C. mai 2001. http://www.w3.org/TR/2001/REC-xmlschema-1-20010502/
P. Biron, A. Malhotra. "XML Schema Part 2: Datatypes". Recommandation W3C. mai 2001. http://www.w3.org/TR/2001/REC-xmlschema-2-20010502/

[XPath] J. Clark, S. DeRose. "XML Path Language (XPath) Version 1.0". Recommandation W3C. octobre 1999. http://www.w3.org/TR/1999/REC-xpath-19991116

[XPointer] S. DeRose, R. Daniel, E. Maler. "XML Pointer Language (XPointer)". Document de travail W3C. janvier 2001. http://www.w3.org/TR/2001/WD-xptr-20010108

[XSL] S. Adler, A. Berglund, J. Caruso, S. Deach, P. Grosso, E. Gutentag, A. Milowski, S. Parnell, J. Richman, S. Zilles. "Extensible Stylesheet Language (XSL)". Proposition de Recommandation W3C. août 2001. http://www.w3.org/TR/2001/PR-xsl-20010828/

[XSLT] J. Clark. "XSL Transforms (XSLT) Version 1.0". Recommandation W3C. novembre 1999. http://www.w3.org/TR/1999/REC-xslt-19991116.html

Adresse des auteurs

Donald E. Eastlake 3rd
Motorola,
20 Forbes Boulevard
Mansfield, MA 02048
USA
téléphone : 1-508-851-8280
mél : Donald.Eastlake@motorola.com

Joseph M. Reagle Jr., W3C
Massachusetts Institute of Technology
Laboratory for Computer Science
NE43-350, 545 Technology Square
Cambridge, MA 02139 USA
téléphone : +1.617.258.7621
mél : reagle@w3.org

David Solo
Citigroup
909 Third Ave, 16th Floor
NY, NY 10043
USA
téléphone : +1-212-559-2900
mél : dsolo@alum.mit.edu

Déclaration de droits de reproduction

Remerciement

Le financement de la fonction d'édition des RFC est actuellement fourni par l'Internet Society.