3. XMLDSig

Uno de los objetivos de nuestro proyecto es la elaboración de una aplicación de firma digital de documentos. Hemos pensado que los documentos que puede manejar nuestra aplicación podrían estar en formato XML (*eXtensible Markup Language*), puesto que se ha convertido en el estándar para almacenar e intercambiar datos a través de Internet.

Así pues, en este capítulo profundizaremos en los conceptos de la firma digital para su inserción en los mensajes XML, que se basa en la especificación XML *Signature Syntax and Processing* (XMLDSig) elaborada por el World Wide Web Consortium o W3C.

Antes que nada, para entender el funcionamiento de XML, en el primer apartado nos adentraremos en el modelo de datos de XML, seguiremos con el estudio de la estructura de un documento y los lenguajes de definición, para terminar la introducción con los lenguajes de consulta.

Es a partir de la sección 4.2. cuando procedemos al estudio del estándar para la inclusión de firmas en formato XML. Estudiamos los tipos de formatos, la estructura y esquema de XMLDSig detallando los elementos que aparecerán como nodos con información del documento XML firmado.

También se detallan los pasos o procesos que hay que seguir conceptualmente para la creación y validación de firmas de la especificación estudiada.

En el apartado 4.8. se explica el porqué de la elección de Java como implementación de XMLDSig. Básicamente si ya hemos conseguido datos independientes de la plataforma con XML, la elección de Java, que también es independiente de ella, cierra el círculo de independencia deseado para nuestra aplicación a desarrollar.

De esta forma, este capítulo sienta las bases teóricas sobre la cual se desarrolla nuestra aplicación de Firma Digital XML, cuyo diseño y particularidades se detallan ya en un capítulo propio.

3.1. Introducción a XML

XML no sólo se utiliza para proporcionar información sobre la estructura y el significado de los datos visualizados en una página web en conjunción con XSL (*eXtensible Stylesheet Language*), que se encarga de cómo se va visualizar la información, sino que se ha propuesto como un posible modelo para el almacenamiento y la recuperación de datos, estando en pleno auge estos sistemas de bases de datos basados en XML.

En esta sección presentaremos el modelo de datos de XML, que está basado en estructuras jerárquicas (árbol), diferentes a las estructuras planas del modelo de datos relacional.

La segunda sección se centra en la estructura de los documentos XML, y en los lenguajes para especificar la estructura de estos árboles, como DTD (*Document Type Definition*) y XML *Schema*. Por último ofrecemos una introducción a los lenguajes de consulta XML: XPath y XQuery.

3.1.1. Modelo de datos jerárquico de XML

En la construcción de un documentos XML se utilizan dos conceptos de estructuración principales: elementos y atributos. Es importante saber que el término atributo en XML no se utiliza de la misma forma que en la terminología de bases de datos, pero sí como se utiliza en los lenguajes de descripción de documentos, como HTML y SGML.

Los atributos en XML proporcionan información adicional que describe los elementos. En XML hay conceptos adicionales, como entidades, identificadores y referencias, pero primero nos centraremos en describir los elementos y los atributos que muestran la esencia de XML.

El siguiente cuadro muestra un ejemplo de documento. Vemos que en XML existe un único elemento raíz y de él anidan los sucesivos.

```
<?xml version= "1.0" encoding="UTF-8" standalone="yes" ?>
<Proyectos>
      <Proyecto>
            <Nombre>Bases de datos XML</Nombre>
            <Área>Telemática</Área>
            <Alumno>
                  <DNI>48935163</DNI>
                  <Nombre>Fernando</Nombre>
                  <Apellido1>Pérez</Apellido1>
                  <Apellido2>Borrero</Apellido2>
            </Alumno>
      </Proyecto>
      <Proyecto>
            <Nombre>Firma digital XML</Nombre>
            <Área>Telemática</Área>
            <Alumno>
                  <DNI>48935163</DNI>
                  <Nombre>Fernando</Nombre>
                  <Apellido1>Pérez</Apellido1>
                  <Apellido2>Borrero</Apellido2>
            </Alumno>
      </Proyecto>
</Proyectos>
```

Código 1: Documento XML *Ejemplo1.xml* con elemento raíz <Proyectos>

Como en cualquier lenguaje de marcas, los elementos están identificados dentro del documento por su etiqueta inicial y su etiqueta final. Los nombres de las etiquetas aparecen encerrados entre los símbolos <...>, y las etiquetas de cierre quedan identificadas además por una barra inclinada </...>.

En este caso, el valor del atributo *standalone* tiene el valor *yes*, lo que significa que el documento es independiente y no se valida contra ninguna DTD o Schema. En la primera línea también aparecen otros atributos como la versión y la codificación del documento.

Muy importante esto último, porque codificaciones diferentes nos harían pensar en principio que estamos ante documentos XML distintos, pero que en realidad contienen información equivalente. Para comprobar eso se usan las funciones de canonización.

Los elementos complejos se construyen jerárquicamente a partir de otros elementos, mientras que los elementos simples contienen valores de datos. Una característica importante de XML es que los nombres de las etiquetas se eligen libremente para describir el significado de los elementos de datos del documento, por lo que es comprensible a simple vista por humanos.

La correspondencia entre la representación textual del anterior documento XML y la estructura en árbol sería la siguiente:

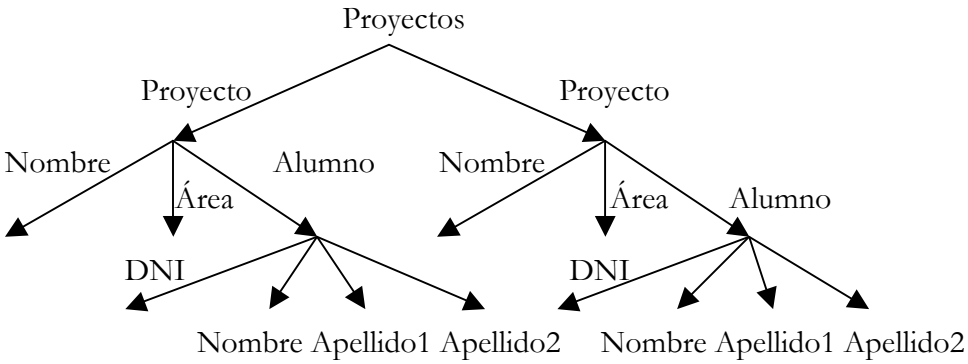


Figura 1: Representación de un documento XML como un grafo

Es por esto por lo que el modelo XML se denomina modelo de árbol o modelo jerárquico. En el código 1, los elementos sencillos son <Nombre> (del proyecto), <Área>, <DNI>, <Nombre> (del alumno), <Apellido1>, <Apellido2>. Los elementos complejos son <Proyectos>, <Proyecto> y <Alumno>. En general, no existe un límite en los niveles de anidamiento de elementos, e incluso pueden llevar el mismo nombre en distintas ramas.
Es posible distinguir tres tipos principales de documentos XML:

- **Documentos centrados en los datos.** Estos documentos tienen muchos elementos de datos pequeños que respetan una estructura específica y, por tanto, pueden extraerse de una base de datos estructurada. Se les da formato como documentos XML para intercambiarlos o visualizarlos por la Web.

- **Documentos XML centrados en el documento.** Son documentos con grandes cantidades de texto, como artículos o libros. En estos documentos hay pocos o ningún elemento de datos estructurado.

- **Documentos XML híbridos.** Estos documentos pueden tener partes que contienen datos estructurados y otras que son principalmente textuales o no estructuradas.

Los documentos XML centrados en los datos se pueden considerar como datos semiestructurados o como datos estructurados. Si un documento XML obedece a un esquema XML predefinido o DTD, entonces el documento puede considerarse como de datos estructurados. Por el contrario, XML permite documentos que no obedecen a ningún

esquema; estos documentos pueden considerarse como datos semiestructurados, que también se conocen como documentos XML sin esquema o independientes (valor del atributo *standalone* es igual a *yes*).

Los atributos en XML se utilizan de una forma parecida a como se utilizan en HTML. Por ejemplo, para describir las propiedades y las características de los elementos (etiquetas) dentro de las que aparecen. En la siguiente sección explicamos los atributos con el uso de DTD y XML *Schema*.

3.1.2. Documentos XML, DTD y XML Schema

3.1.2.1. Documentos bien formados y válidos

Se dice que un documento XML está bien formado si cumple unas serie de condiciones:

- Debe empezar con una declaración XML para indicar la versión XML que se está usando, así como cualesquiera otros atributos relevantes, como se muestra en la primera línea del código 1.

- También debe obedecer a la directrices sintácticas del modelo en árbol. Esto significa que sólo debe haber un elemento raíz en el documento, y que todos los elementos deben incluir un par de etiquetas inicial y final coincidentes, dentro de las etiquetas de inicio y cierre del elemento padre. Estas condiciones son las necesarias y suficientes para que los elementos anidados constituyan una estructura en árbol bien formada.

Un documento XML bien formado es sintácticamente correcto. Esto permite que sea procesado por los procesadores genéricos que recorren el documento y crean una representación interna en forma de árbol.

Un conjunto estándar de funciones (API) denominado DOM (*Document Object Model*) permite a los programas manipular la representación en árbol resultante correspondiente a un documento XML bien formado. No obstante, es preciso analizar de antemano el documento entero usando DOM. Otra API, denominada SAX, permite el procesado de documentos XML sobre la marcha notificando al programa de procesamiento siempre que se encuentra la etiqueta de inicio o cierre correspondiente.

Un fichero XML bien formado puede tener cualquier nombre de etiqueta para los elementos del documento. No hay un conjunto predefinido de elementos (nombres de etiqueta) que un programa que procesa el documento deba esperar encontrarse. Gracias a ello, el creador del documento tiene libertad para especificar elementos nuevos, sin embargo, se limita las posibilidades de interpretar automáticamente los elementos del documento.

Un criterio más estricto que un documento XML debe cumplir es el de la validez. En este caso, el documento debe estar bien formado, y los nombres de los elementos utilizados en los pares de etiquetas de inicio y fin deben seguir la estructura especificada en un archivo DTD (*Document Type Definition*) separado o en un archivo de esquema XML.

Vamos a explicar primero XML DTD y después ofreceremos una visión general de XML *Schema*, ya que es bastante amplio.

3.1.2.2. XML DTD

A continuación se muestra una DTD sencilla, que especifica los elementos (nombres de etiqueta) y sus estructuras anidadas de acuerdo al código 1.

```
<!DOCTYPE Proyectos [
   <!ELEMENT Proyectos (Proyecto+)>
   <!ELEMENT Proyecto (Nombre, Área, Alumno+)>
   <!ELEMENT Nombre (#PCDATA)>
   <!ELEMENT Área (#PCDATA)>
   <!ELEMENT Alumno (DNI, Nombre, Apellido1, Apellido2?)>
   <!ELEMENT DNI (#PCDATA)>
   <!ELEMENT Nombre (#PCDATA)>
   <!ELEMENT Apellido1 (#PCDATA)>
   <!ELEMENT Apellido2 (#PCDATA)>
]>
```

Código 2: XML DTD *Proyectos.dtd*

Se utiliza la siguiente notación:

Sufijo	Tipo	Significado
?	Elemento (no repetitivo) monovalor opcional	El elemento puede tener uno o ninguno de los hijos especificados, pero nunca más de uno
*	Elemento (repetitivo) multivalor opcional	El elemento puede tener cero o más de los hijos especificados
+	Elemento (repetitivo) multivalor obligatorio	El elemento debe tener al menos uno de los hijos especificados, pero puede tener más

Un elemento que aparece sin ninguno de los tres símbolos anteriores debe aparecer exactamente una vez en el documento. Se les conoce como elementos (no repetitivo) monovalor obligatorio.

El tipo de elemento se especifica mediante unos paréntesis a continuación del elemento. Si los paréntesis incluyen nombres de otros elementos, estos últimos son los hijos del elemento en la estructura en forma de árbol. Si los paréntesis incluyen la palabra clave #PCDATA o uno de los otros tipos de datos disponibles en DTD, el elemento es un nodo hoja (elemento sencillo).

Podemos ver que la estructura en árbol de la figura 1 y el documento XML del código 1 son conformes al XML DTD del código 2. Para forzar la comprobación de la conformidad de un documento XML con un DTD, debemos declararlo en el documento. Por ejemplo, podríamos modificar la primera línea del código 1 por lo siguiente:

```
<?xml version= "1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE Proyectos SYSTEM "proyectos.dtd">
```

Como hemos visto, una DTD especifica la clase de documento XML dictando una estructura. En general:

- Describe un formato de datos.

    - Elementos: cuáles son las etiquetas permitidas y cuál es el contenido de cada etiqueta.
    - Estructura: en qué orden van las etiquetas en el documento.
    - Anidamiento: qué etiquetas van dentro de cuáles.

- Usa un formato común de datos entre aplicaciones.

- Verifica los datos al intercambiarlos.

- Verifica un mismo conjunto de datos.

Los elementos de una DTD son los siguientes:

- Elementos con contenido ELEMENT. Un elemento tiene contenido ELEMENT, si solo puede contener a otros elementos, opcionalmente separados por espacios en blanco.

- Elementos con contenido TEXT. Un elemento tiene contenido TEXT, si solo puede contener texto o (PCDATA = printable character data).

- Elementos con contenido MIXED. Un elemento tiene contenido MIXED, si puede contener texto u otros elementos.

- Elementos con contenido EMPTY. Un elemento tiene contenido EMPTY, si no puede contener otros elementos.

Algunos de los atributos para DTD los siguientes:

- CDATA: texto (*Character Data*)
- NMTOKEN: "abc...z0123..9-_:."
- NMTOKENS: NMTOKEN + espacios

Aunque la DTD es adecuada para definir estructuras en árbol con elementos obligatorios, opcionales y repetitivos, tiene varias limitaciones. En primer lugar, los tipos de datos en DTD sólo son PCDATA (texto plano). En segundo lugar, DTD tiene su propia sintaxis especial y, por tanto, requiere procesadores diferentes que los usados para analizar documentos XML. En tercer lugar, todos los elementos DTD están obligados a seguir siempre el orden especificado del documento, por lo que no están permitidos los elementos desordenados. Estos inconvenientes llevaron al desarrollo de XML *Schema*, un lenguaje más general para especificar la estructura y los elementos de los documentos XML.

3.1.2.3. XML Schema

Utiliza las mismas reglas sintácticas que los documentos XML, por lo que pueden utilizarse los mismos procesadores. Para distinguir los dos tipos de documentos, utilizaremos el término documento de instancia XML (documento XML) y documento de esquema XML para uno que especifica un esquema XML.

El esquema del código 3 serviría para el propósito de definir la estructura de una base de datos si se almacenara en un sistema de XML nativo. Explicamos este tema en el apartado correspondiente del siguiente bloque; Documentos XML y bases de datos.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="Proyectos">
    <xs:complexType>
      <xs:sequence>
        <xs:element maxOccurs="unbounded" ref="Proyecto"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="Proyecto">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="Nombre"/>
        <xs:element ref="Área"/>
        <xs:element ref="Alumno"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="Área" type="xs:NCName"/>
  <xs:element name="Alumno">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="DNI"/>
        <xs:element ref="Nombre"/>
        <xs:element ref="Apellido1"/>
        <xs:element ref="Apellido2"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="DNI" type="xs:integer"/>
  <xs:element name="Apellido1" type="xs:NCName"/>
  <xs:element name="Apellido2" type="xs:NCName"/>
  <xs:element name="Nombre" type="xs:NCName"/>
</xs:schema>
```

Código 3: XML *Schema Proyectos.xsd*

Como en DTD, *Schema* está basado en el modelo de datos en árbol, con los elementos y los atributos como los conceptos de estructuración principales. Sin embargo, toma prestados conceptos adicionales de los modelos de bases de datos y de objetos, como las claves, las referencias y los identificadores. A continuación describimos por pasos, siguiendo el orden de utilización en el anterior esquema, las características de XML *Schema*.

1. **Descripción de esquema y espacios de nombres.** Es necesario identificar el conjunto específico de elementos del lenguaje XML *Schema* (etiquetas) que se utilizan definiendo un archivo almacenado en una ubicación web. La segunda línea del código 3 especifica el archivo usado en este ejemplo, que es `http://www.w3.org/2001/XMLSchema`. Es un estándar que se invoca con frecuencia en XML *Schema*. Toda definición semejante se denomina espacio de nombres XML, porque define el conjunto de comandos (nombres) que pueden usarse. El nombre de archivo se asigna a la variable `xs` (descripción de XML Schema) utilizando el atributo `xmlns` (espacio de nombres XML), y esta variable se

utiliza como prefijo para todos los comandos de XML *Schema*. Por ejemplo, cuando usamos xs:element o xs:sequence, nos estamos refiriendo a las definiciones del elemento y la secuencia de etiquetas tal y como se definen en `http://www.w3.org/2001/XMLSchema`.

2. **Elementos y tipos.** A continuación, especificamos el elemento raíz. El atributo `name` de la etiqueta `xs:element` define el nombre del elemento, llamado Proyectos, para el elemento raíz de nuestro documento XML. La estructura del elemento raíz Proyectos puede especificarse después, en nuestro ejemplo es `xs:complexType`.

3. **Elementos de primer nivel y sucesivos en la base de datos.** A continuación especificamos los elementos de primer nivel bajo el elemento raíz `Proyectos`. En nuestro caso ese primer nivel está formado por una secuencia de un único tipo de elemento llamado `Proyecto`. Igualmente que para `Proyectos`, se especifica con una etiqueta `xs:element`. Si una etiqueta sólo tiene atributos y ningún subelemento o dato en su interior, puede darse por finalizada directamente con el símbolo (/>) en lugar de tener la etiqueta de cierre correspondiente. Son conocidos como elementos vacíos; ejemplo de ello es el elemento `Proyecto`. Para los elementos hijos de `Proyecto`, se define una secuencia (los elementos deben aparecer en ese orden) de `Nombre`, `Área` y `Alumno` usando de nuevo la etiqueta `xs:sequence`. De nuevo, para definir cada elemento usamos la estructura `xs:element` de tal forma que podamos especificar todos los tipos posibles de elementos que nos encontraremos en el documento XML. El elemento `Nombre` es un tipo especial de elemento, ya que puede aparecer dentro de `Proyecto` o de `Alumno`, por lo que lo hemos esperado a definido en último lugar.

4. **Declarar el tipo de elemento y las ocurrencias mínima y máxima.** Los atributos `maxOccurs`, `minOccurs` y `Type` de la etiqueta `xs:element` especifican la multiplicidad de las etiquetas y el tipo de cada elemento conforme a las especificaciones del esquema. El tipo NCName es un tipo derivado del tipo simple String en XML *Schema*. Las etiquetas `maxOccurs` y `minOccurs` se utilizan para acotar los límites inferior y superior del número de apariciones de un elemento en cualquier documento. En nuestro caso no especificamos `minOccurs`. En estos casos lo predeterminado es exactamente una ocurrencia. Es parecido a los símbolos ?, * y + en DTD.

	minOccurs	maxOccurs	Equivalente DTD
Una sola vez	1	1	elemento
Una o más veces	1	unbounded	elemento+
Cero o una vez	0	1	elemento?
Cero o más veces	0	unbounded	elemento*
No debe aparecer	0	0	-elemento

5. **Los elementos globales.**

`<xs:element name="Área" type="xs:NCName"/>` es un ejemplo de elemento global. Cumple lo siguiente:

- tiene que estar declarado directamente como un subelemento del elemento `<xs:schema>`, pero nunca como parte de un elemento de tipo complejo `<xs:complexType>`.
- no puede contener referencias, es decir, no pueden albergar el atributo `ref`.
- no puede indicar el número de ocurrencias de un elemento.

`<xs:element maxOccurs="unbounded" ref="Proyecto"/>` es la sintaxis de una declaración local que hace referencia de un elemento global

- el atributo `ref` hace referencia a un elemento global.
- en la declaración local si queremos podemos indicar la cardinalidad del elemento global con los atributos `minOccurs` y `maxOccurs`.

El ejemplo de código 3 ilustra algunas de las características principales de XML *Schema*. Hay otras muchas, pero quedan fuera del objetivo de nuestro proyecto.

3.1.3. Consulta XML

Hay varias propuestas como lenguajes de consulta XML, pero destacan dos estándares. El primero es XPath, que proporciona estructuras de lenguaje para especificar las expresiones de ruta que permitan identificar ciertos nodos (elementos) dentro de un documento XML que coincidan con patrones específicos. El segundo es XQuery, que es un lenguaje de consulta más general.

XQuery utiliza expresiones XPath, pero ofrece construcciones adicionales. Otros lenguajes como XUpdate, que no es un estándar amparado por el W3C, fueron desarrollados para cubrir las desventajas de XQuery, pero desafortunadamente fueron proyectos abandonados hace tiempo, y cuya última versión data del año 2000.

Esta sección ofrecemos una visión general de XPath y XQuery.

3.1.3.1. XPath

Una expresión XPath devuelve una colección de nodos elemento que satisfacen los patrones especificados en la expresión. Los nombres que aparecen en la expresión XPath son nombres de nodo del árbol del documento XML, que son los nombres de las etiquetas (elemento) o los nombres de los atributos, posiblemente con condiciones de calificador adicionales para restringir después los nodos que cumplen el patrón.

Al especificar una ruta se suelen utilizar dos separadores: una sola barra inclinada (/) y una barra inclinada doble (//). Una única barra inclinada delante de una etiqueta indica que ésta debe aparecer como un hijo directo de la etiqueta anterior (padre), mientras que una barra inclinada doble hace referencia a que la etiqueta puede aparecer como un descendiente de cualquier nivel de la etiqueta anterior.

A continuación ofreceremos algunos ejemplos de XPath de acuerdo al esquema XML del código 3.

1. `/Proyectos`

2. `/Proyectos/Proyecto`

3. `//Alumno/Nombre`

4. `/Proyectos/Proyecto/Alumno/Nombre`

5. `/Proyectos/Proyecto [Área e Telemática]`

La primera expresión XPath devuelve el nodo raíz Proyectos y todos sus nodos descendientes, es decir, devuelve el documento XML entero.

Debemos saber que es costumbre incluir el nombre del fichero o recurso en la consulta XPath. Esto nos permite hacer referencia a cualquier nombre de fichero local o, incluso, cualquier ruta de acceso correspondiente a un fichero en la web.

Por ejemplo, si el documento XML *proyectos.xml* está almacenado en la siguiente ubicación:

```
www.servidor.com/proyectos.xml
```

Esa primera expresión puede rescribirse como:

```
doc(www.servidor.com/proyectos.xml)/Proyectos
```

Este prefijo podría incluirse igualmente en los demás ejemplos.

El segundo ejemplo, devuelve todos los nodos de `Proyecto` y sus subárboles descendientes. Los nodos (elementos) de un documento XML están ordenados, por lo que el resultado de XPath que devuelva varios nodos lo hará siguiendo el mismo orden en que aparecen los nodos en el árbol del documento.

La tercera sentencia ilustra el uso de //, que es conveniente usar si no conocemos la ruta de acceso completa que estamos buscando, pero conocemos el nombre de algunas etiquetas de interés dentro del documento XML. Esto es particularmente útil para los documentos XML sin esquema o para los documentos con muchos niveles de nodos anidados.

La expresión devuelve todos los nodos `Nombre` que son hijos directos de un nodo `/Alumno`, por lo que no se devolverán los nodos `Nombre` del elemento padre `Proyecto`.

El cuarto ejemplo debe devolver el mismo resultado que el anterior, ya que esta vez hemos escrito la ruta de acceso completa.

La última expresión entrega todos los nodos `Proyecto` y sus nodos descendientes que son hijos y que están bajo una ruta de acceso `/Proyectos/Proyecto` y tienen un nodo hijo `Área` con el valor `Telemática`.

XPath tiene varias operaciones de comparación que podemos utilizar en las condiciones de calificador, incluyendo operaciones estándar para cálculos aritméticos, manipulación de cadenas y comparación de conjuntos.

3.1.3.2. XQuery

Hemos visto que XPath permite escribir expresiones que seleccionan nodos a partir de un documento XML estructurado en forma de árbol.

XQuery permite especificar consultas más generales sobre uno o más documentos XML. La forma típica de una consulta en XQuery se conoce como expresión FLWR, que hace referencia a las cuatro cláusulas principales de XQuery y que tienen la siguiente forma:

FOR <asociaciones de variables a nodos (elementos) individuales>

LET <asociaciones de variables a colecciones de nodos (elementos)>

WHERE <condiciones de calificador>

RETURN <especificación del resultado de la consulta>

A continuación ofreceremos algunos ejemplos que se pueden declarar en documentos de instancia XML de acuerdo al documento de esquema XML del código 3.

1. FOR $x IN
   `doc(www.servidor.com/proyectos.xml)`
   `//Alumno [DNI gt 40000000]`
   RETURN <res> $x`/Apellido1`, $x`/Apellido2` </res>

2. FOR $x IN
   `doc(www.servidor.com/proyectos.xml)/Proyectos/Proyecto/Alumno`
   WHERE $x/DNI gt 40000000
   RETURN <res> $x`/Apellido1`, $x`/Apellido2` </res>

3. FOR $x IN
   `doc(www.servidor.com/proyectos.xml)/Proyectos/Proyecto`
   $y IN `doc(www.servidor.com/proyectos.xml)/Proyectos/Proyecto /Alumno`
   WHERE $x`/Área e Telemática` AND $y`/Alumno/Nombre` e Fernando
   RETURN <res> $x`/Nombre` </res>

La primera de las consultas recupera los dos apellidos del alumno con DNI mayor que cuarenta millones. La variable $x está asociada a cada elemento `Alumno`, pero sólo para aquellos elementos que satisfacen el calificador ya mencionado. El resultado recupera los elementos hijo `Apellido1` y `Apellido2` de los elementos `Alumno` que hayan sido seleccionados.

El segundo ejemplo es una forma alternativa de recuperar los mismos elementos que recupera la primera consulta.

La última consulta nos muestra cómo puede se realizar la operación de concatenación con más de una variable. Aquí la variable $x está asociada a cada elemento `Proyecto` y la variable $y se asocia a cada elemento `Alumno`. El objetivo va a ser recuperar el nombre de todos los proyectos realizados en el Área de Telemática por alguien cuyo nombre sea Fernando.

3.2. Introducción a XMLDSig

La firma digital en XML se realiza a nivel de documento (mensaje), es decir, se firma todo el mensaje XML. Además un documento puede ser firmado por más de una persona, pero se debe cumplir que sólo exista un firmante por firma. Además existe la posibilidad de tener varios elementos firmados por firma y de diferentes tipos.

El algoritmo de firma que se propone es RSA, que se apoya en el algoritmo de *hash* SHA-1 para el cálculo del valor de resumen mientras que los certificados X.509v3 de la Organización Internacional de Estándares son el formato escogido por el W3C.

El estándar define diversas formas de crear documentos firmados diferenciándose entre el formato *Enveloped Signature, Enveloping Signature* y *Detached Format.*

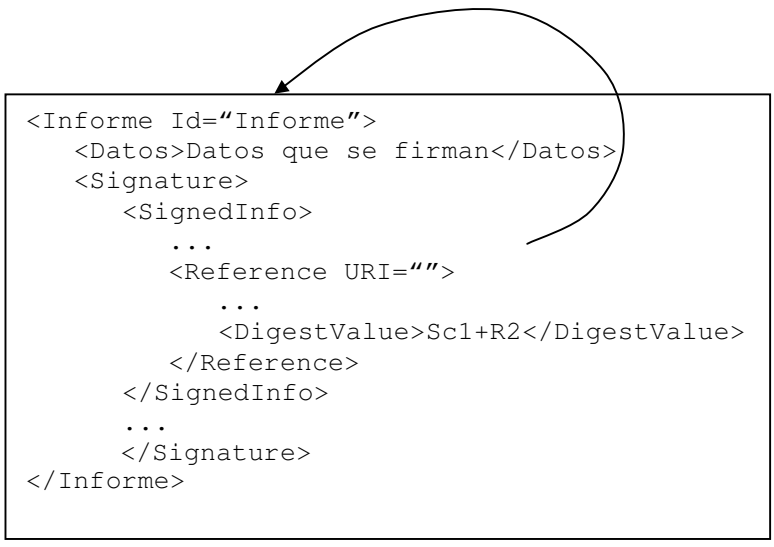
Las diferencias entre estos formatos se encuentran en la posición en la que se coloca la firma respecto del objeto firmado (forma de hacer las referencias), lo que dará lugar a aplicar diferentes transformaciones al objeto si queremos adaptarnos a un tipo de formato u otro.

3.3. Formatos de firma XMLDsig

3.3.1. Enveloped Signature

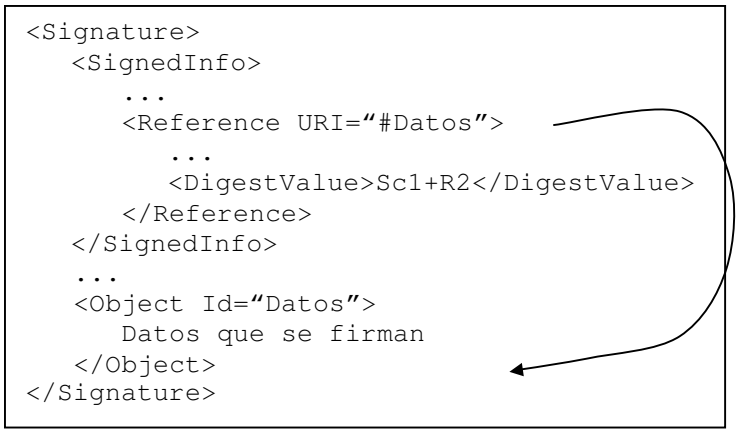
En el tipo de firma *Enveloped Signature* la característica principal es que la firma se encuentra dentro del objeto firmado. Se dice que la firma es hija.

El resultado de un documento firmado será un elemento firma (<Signature>) que se incluye como último elemento en el documento XML.



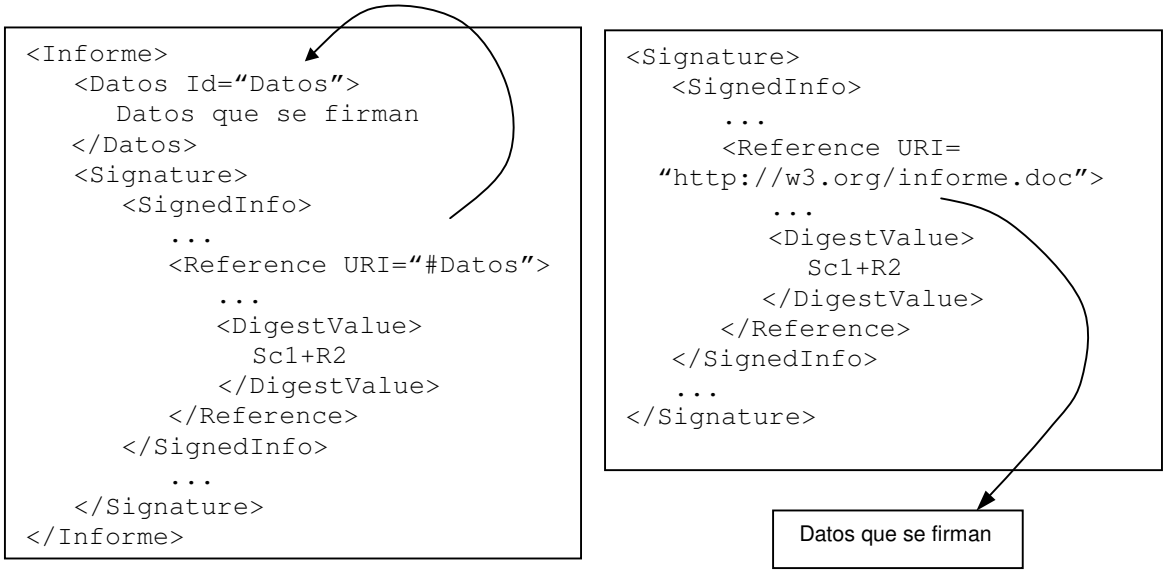
3.3.2. Enveloping Signature

Para este formato el objeto firmado se encuentra dentro de la firma. En este caso se dice que la firma es padre.



3.3.3. Detached Format

Aquí la característica principal es que la firma y el objeto firmado son independientes. Se hace referencia a un recurso externo de red.



3.4. Estructura y esquema de XMLDSig

El espacio de nombres XML (XML-ns) URI (*Uniform Resource Identifiers*) que se debe usar para implementar la inclusión de la firma digital es:

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

Este espacio de nombres (*namespace*) se usa también como prefijo para los identificadores de los algoritmos y recursos.

La firma digital se aplica arbitrariamente sobre cualquier tipo de datos mediante una referencia a un elemento o varios de ellos que contienen la información que debe firmarse.

El resultado de la firma, los algoritmos de *hash* usados, las referencias y los algoritmos de firma se estructuran bajo el elemento <Signature>, con la siguiente estructura:

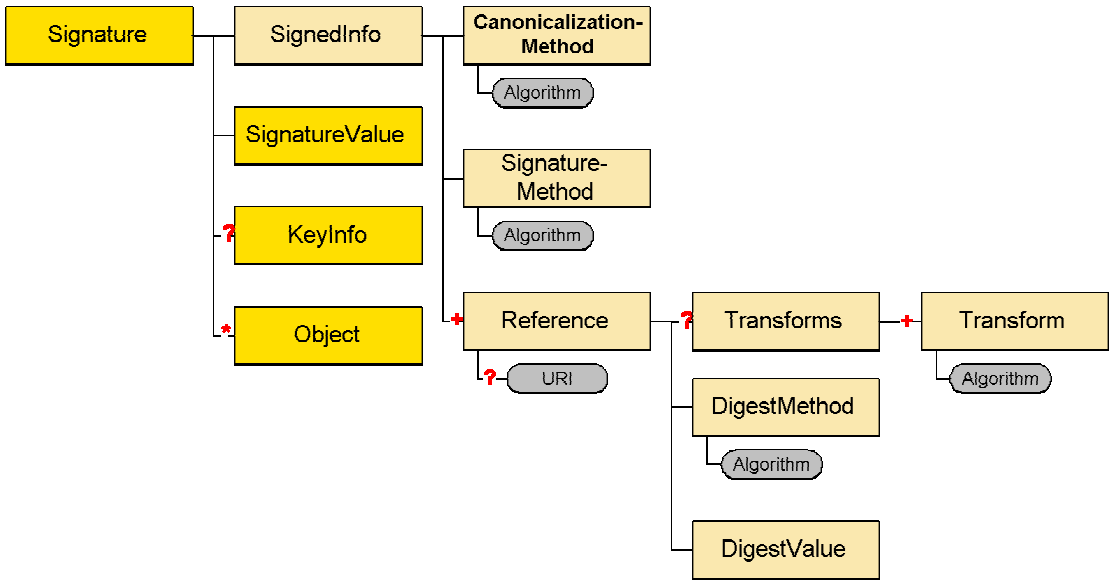
```
<Signature ID?>
   <SignedInfo>
      <CanonicalizationMethod/>
      <SignatureMethod/>
      (<Reference URI? >
         (<Transforms>)?
         <DigestMethod>
         <DigestValue>
      </Reference>)+
   </SignedInfo>
   <SignatureValue>
   (<KeyInfo>)?
   (<Object ID?>)*
</Signature>
```

```
? denota 0 ó 1 ocurrencia

+ denota 1 ó más
ocurrencias

* denota 0 ó más
```

El siguiente esquema es el propuesto para el uso de la especificación XMLDSIG:



Como podemos observar en la figura anterior, se puede dividir la estructura de la firma digital o el elemento `<Signature>` en los siguientes elementos:

- `<SignedInfo>`
- `<SignatureValue>`
- `<KeyInfo>`
- `<Object>`

A continuación veremos con detalle cada uno de estos elementos.

3.4.1. Elemento <SignedInfo>

El elemento <SignedInfo> incluye la información de cómo y qué se firma.

En él se muestra qué algoritmos de canonización y firma (<CanonicalizationMethod>-<SignatureMethod>), serán utilizados para obtener el contenido del elemento <SignatureValue>. Además puede tener las referencias a los objetos que se firmarán mediante el elemento <Reference>. Para cada uno de los objetos firmados éste elemento establece una referencia al objeto de datos de interés.

Por tanto lo que indica este elemento es lo que realmente se firma.

3.4.1.1. Elemento <CanonicalizationMethod>

En este elemento iniciamos el algoritmo utilizado para canonizar el elemento de <SignedInfo> antes de realizar el *hash,* durante la creación de la firma.

El algoritmo de canonización utilizado se indica en el atributo *Algorithm*. En el caso de usar el tipo Exclusive Canonical XML se tiene:

```
Algorithm="http://www.w3.org/TR/xml-exc-c14n"
```

Existen multitud de métodos que recomienda el W3C entre ellos:

- Required Canonical XML (omits comments)

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

- Recommended Canonical XML with Comments

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

3.4.1.2. Elemento <SignatureMethod>

Este elemento especifica el algoritmo que debe usarse para generar la firma a partir de la versión canonizada de <SignedInfo>.

El resultado de la firma debe indicarse en el elemento <SignatureValue>.

El proceso de la firma combina una función de *hash* junto con un algoritmo de firma dependiente de la clave.

El algoritmo de firma utilizado se indica en el atributo *Algorithm*. Si usamos RSA-SHA1:

```
Algorithm="http://www.w3.org/2000/09/XMLDSig#rsa-sha1"
```

Como este elemento se incluye en el elemento <SignedInfo> también se firma.

3.4.1.3. Elemento <Reference>

En este elemento se incluye una referencia al objeto que se firmará a través de un identificador URI, que podrá cambiar su sintaxis en función del formato de firma que usemos.

Dentro del objeto <Reference> se encuentra la descripción del algoritmo de *hash* utilizado para calcular el valor de resumen y el resultado de aplicar una función de canonización y un algoritmo de *hash* a ese objeto. Las "marcas" asociadas a estos valores son <DigestMethod> y <DigestValue> respectivamente.

Por último el elemento <Reference> también puede tener asociada transformaciones (<Transforms>), que se aplicarán sobre el objeto referenciado y la salida de éstas es entregada a la función de *hash* que se esté ocupando.

Los siguientes ejemplos muestran lo que el atributo URI identifica y cómo es referenciado el objeto de datos a firmar según el formato de la firma:

URI="http://example.com/bar.xml"
: Identifica a los bytes que representan el recurso externo "http://example.com/bar.xml", que es probablemente un documento XML dada la extensión del archivo.

URI="http://example.com/bar.xml#chapter1"
: Identifica al elemento con el valor del atributo ID "chapter1" del recurso externo XML "http://example.com/bar.xml", que se proporciona como un flujo de bytes.

URI=""
: Identifica el nodo origen (node-set) del recurso XML que contiene la firma.

URI="#chapter1"
: Identifica un nodo origen que contiene el elemento con el valor del atributo ID "chapter1" del recurso XML que contiene la firma. La firma XML modifica este nodo para incluir el elemento más todos los descendentes incluyendo espacios de nombres y atributos, pero no comentarios.

Si se tratase del tipo *Enveloped Signature*, la información a firmar es la etiqueta padre del documento. Así la URI tiene la sintaxis: *URI=""*

3.4.1.3.1. Elemento <Tranforms>

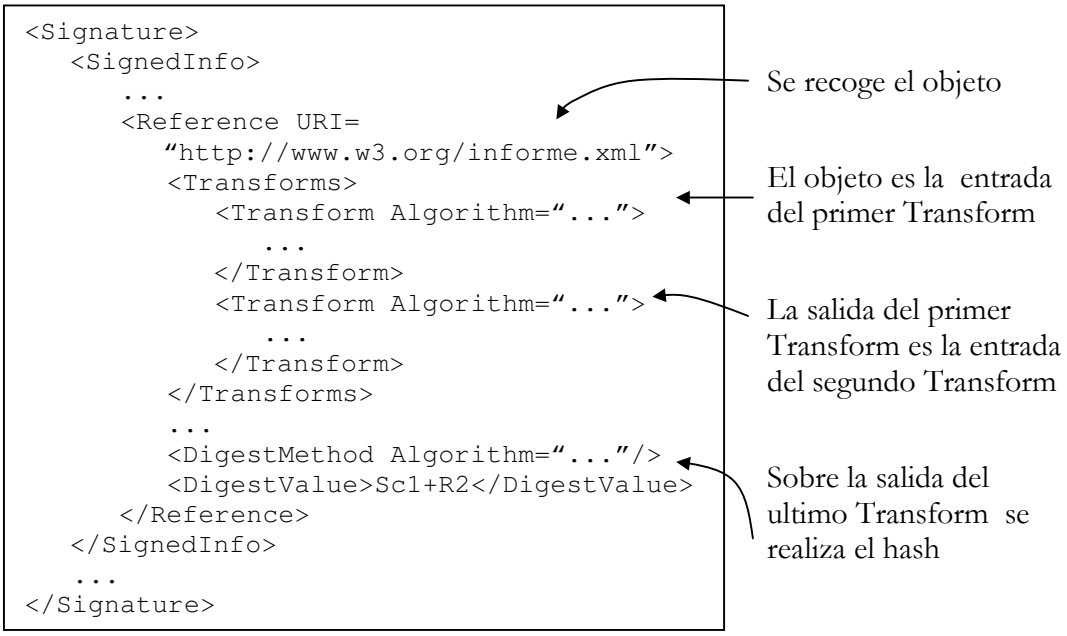
Este elemento opcional contiene la lista de transformaciones que aplicamos al objeto referenciado por el atributo URI del elemento <Reference>, antes de calcular su *hash*.

La lista describe cómo el firmante obtuvo el objeto de datos que se trató.

Se realiza mediante sucesivas repeticiones del elemento hijo <Transform>. Cada transformación consiste en un atributo *Algorithm*.

La salida de cada transformación sirve como entrada para la siguiente. La entrada a la primera transformación viene dada por la referencia del atributo URI del

elemento <Reference>. La salida de la última transformación es la entrada para el algoritmo que se indica en <DigestMethod>.




```
<Signature>
   <SignedInfo>
      ...
      <Reference URI=
         "http://www.w3.org/informe.xml">
         <Transforms>
            <Transform Algorithm="...">
               ...
            </Transform>
            <Transform Algorithm="...">
               ...
            </Transform>
         </Transforms>
         ...
         <DigestMethod Algorithm="..."/>
         <DigestValue>Sc1+R2</DigestValue>
      </Reference>
   </SignedInfo>
   ...
</Signature>
```

Hasta ahora hemos descrito cómo se realizan las transformaciones, pero ¿qué se consiguen con ellas?. La respuesta es manipular los datos antes de realizar el *hash*. De esta forma se puede conseguir:

- Decodificación base64 (MIME)
- Canonización (XML-C14N)
- Transformaciones XSL (XSLT)
- Separación de la firma en el caso Enveloped Signature.
- Selección de una parte de un documento XML, para firmar partes de un documento por separado.

Por ejemplo, si aplicamos la transformación Enveloped Signature seguida de una canonización la lista de transformaciones aplicadas serán:

- Transformación Enveloped Signature:

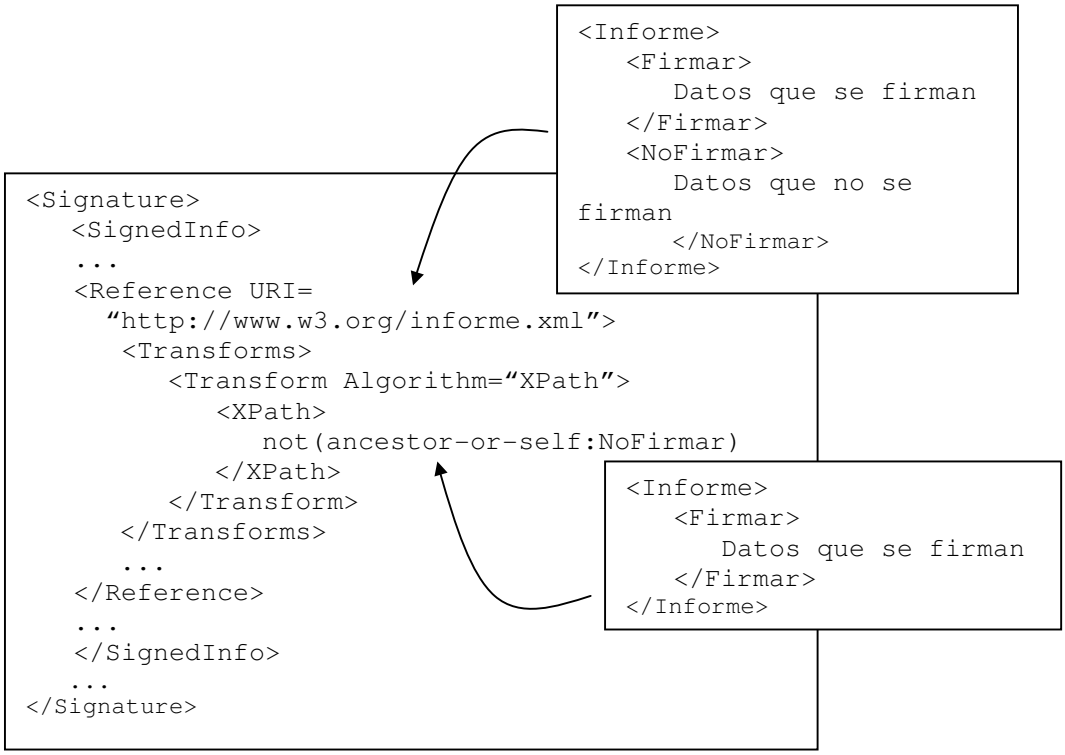
```
Algorithm="http://www.w3.org/2000/09/XMLDSig#enveloped-
signature"
```

- Exclusive Canonical XML

```
Algorithm="http://www.w3.org/TR/xml-exc-c14n"
```

La transformación Enveloped Signature consigue la eliminación del elemento <Signature> del cálculo de la firma cuando la firma está dentro del contenido (objeto) que va a ser firmado evitándose que la firma pase por sucesivas transformaciones que puedan modificar su valor. También se elimina este elemento al validar la firma.

Si lo que queremos es firmar partes de un documento debemos utilizar una transformación XPath.



Como puede imaginarse la transformación Enveloped Signature puede ser implementada alternativamente con la transformación XPath, ya que lo que se busca es aislar de la firma una parte del documento XML. En nuestro caso el elemento que se excluye sería `<Signature>`.

Se puede dar el caso de que información XML equivalente pueda diferir en su representación. Los motivos pueden ser los siguientes:

- Orden de atributos, valores por defecto.
- Referencias a entidades.
- Utilización de espacios de nombres diferentes.
- Pérdida de información por la utilización de analizadores sintácticos XML como SAX o DOM, que nos permiten recorrer los datos XML de principio a fin y procesar la estructura de datos que soporta un documento XML.

Para solucionar esto surge la canonización, que se encarga de calcular la forma canónica de un documento XML garantizando la integridad de los datos.

Si queremos obtener los datos XML referenciados por el elemento `<Reference>` debemos añadir una transformación Canonical XML al elemento `<Reference>` como se indica en el esquema de la página siguiente:

```
<Signature>
   <SignedInfo>
      ...
      <Reference URI="http://www.w3.org/informe.xml">
         <Transforms>
            <Transform URI="Canonical XML">
            </Transform>
         </Transforms>
         ...
      </Reference>
      ...
   </SignedInfo>
   ...
</Signature>
```

3.4.1.3.2. Elemento `<DigestMethod>`

Este elemento especifica el algoritmo de *hash* (entiéndase también como algoritmo de *digest* o resumen) utilizado para calcular el resumen cuando el objeto de datos referenciado en el atributo *URI* del elemento `<Reference>` ha sido canonizado. El resultado de este resumen se pone en el elemento `<DigestValue>`.

Los algoritmos de *digest* soportados son:

- Required SHA1

```
Algorithm="http://www.w3.org/2000/09/XMLDSig#sha1"
```

3.4.1.3.3. Elemento `<DigestValue>`

`<DigestValue>` es el elemento que contiene el valor resultante de aplicar el algoritmo especificado en el atributo *Algorithm* del elemento anterior. El resultado se codifica en el formato base64 (ver apartado 2.5).

3.4.2. Elemento `<SignatureValue>`

El elemento `<SignatureValue>` contiene el resultado de la firma digital al ser aplicada al elemento `<SignedInfo>`. El valor de la firma se codifica siempre en base64. Posee un atributo `Id` que debe ser único, ya que será lo que identifique a la firma en posteriores procesos de validación.

3.4.3. Elemento `<KeyInfo>`

El elemento opcional `<KeyInfo>` contiene información sobre la clave pública (o varias de ellas) que será utilizada para validar la firma, certificados asociados a esa clave u otra información.

Los valores que contenga este elemento dependerán de la forma de trabajo adoptada. Para ser más específicos, si consideramos el caso en que una organización valida todas las firmas de acuerdo con la información de certificados disponibles en una base de datos de la organización, el elemento `<KeyInfo>` puede ser obviado porque no presentará contenido.

3.4.3.1. Elemento `<KeyName>`

Normalmente contiene un identificador de tipo cadena relacionado con la pareja de claves utilizada para firmar el mensaje aunque puede contener información relacionada con el protocolo que indirectamente identifique la pareja de claves.

3.4.3.2. Elemento `<KeyValue>`

Especifica la clave para validar la firma digital.

Los formatos estructurados para definir las claves públicas DSA (REQUIRED) y RSA (RECOMMENDED) son:

- Required DSAwithSHA1 (DSS)

Algorithm="http://www.w3.org/2000/09/XMLDSig#dsa-sha1"

- Recommended RSAwithSHA1

Algorithm="http://www.w3.org/2000/09/XMLDSig#rsa-sha1"

3.4.3.2.1. Elemento `<DSAKeyValue>`

Las claves DSA y el algoritmo de firma DSA se incluyen en este elemento. Están especificados en la norma DSS. Los valores de las claves públicas DSA pueden tener los siguientes campos:

P Número primo de longitud entre 512 y 1024 bits (múltiplo de 64 bits)

Q Número primo que un divisor de p-1 y tiene una longitud de 160 bits

G $G=h^{(p-1)/Q}$ mod p, donde $1<h<p-1$ y G es mayor que 1

Y $Y=G^{x}$ mod p, donde X es un entero aleatorio mayor que 0 y menor que Q

J J=(p-1)/Q Se usa para mejorar la eficiencia en el cálculo de G

seed semilla para generar el número primo DSA

pgenCounter contador para generar el número primo DSA

Los dos últimos parámetros son opcionales, y en caso de aparecer, deben hacerlo conjuntamente.

Los valores de P, Q y G son públicos. La clave pública es Y y la privada es X. Todos los parámetros son codificados en base 64 mientras que los enteros de longitud variable se representan mediante cadenas de octetos (*octect strings)* definidas por el tipo CryptoBinary.

3.4.3.2.2. Elemento `<RSAKeyValue>`

Los valores de las claves RSA se especifican con dos elementos: el módulo y el exponente.

`<Modulus>` Contiene el valor del módulo en base 64

`<Exponent>` Contiene el valor del exponente en base 64

3.4.3.3. Elemento `<RetrievalMethod>`

Se usa para transmitir una referencia a la información de `<KeyInfo>` que ha sido guardada en otra localización.

Por ejemplo, varias firmas en un documento pueden usar una clave verificada por una cadena de certificados X.509v3 apareciendo una vez en el documento o de forma remota a él. Cada firma de `<KeyInfo>` puede referenciar esta cadena usando un único elemento `<RetrievalMethod>` en lugar de incluir la cadena completa con una secuencia de elementos `<X509Certificate>` que se verán en el siguiente apartado.

3.4.3.4. Elemento `<X509Data>`

Contiene una referencia al certificado asociado al usuario y su clave mediante el elemento `<X509IssuerSerial>`, o en su defecto el certificado mismo mediante `<X509Certificate>` o la CRL (Certificate Revocation List) mediante el elemento `<X509CRL>`.

Por lo tanto sólo puede aparecer uno de estos elementos, aunque también existen otros como `<X509SubjectName>` o `<X509SKI>` que hacen referencia al certificado que contiene la clave para la validación y que también aparecen por separado.

`<X509IssuerSerial>` se compone de dos elementos hijos:

`<X509IssuerName>` es el nombre del que expide el certificado

`<X509SerialNumber>` es el número de serie

3.4.3.5. Elemento `<PGPData>`

Se usa para transmitir la información relacionada con el par de claves públicas PGP y las firmas en dichas claves. Se compone de dos elementos hijos:

`<PGPKeyID>` identificador de una clave pública PGP estándar en base64

`<PGPKeyPacket>` es un paquete material de claves en base64

3.4.3.6. Elemento `<SPKIData>`

Se usa para transmitir la información relacionada con el par de claves públicas SPKI, certificados y otros datos SPKI. El elemento hijo `<SPKISexp>` debe aparecer al menos una vez y representa la codificación base64 de una expresión canónica SPKI.

3.4.3.7. Elemento `<MgmtData>`

Es un valor de tipo *string* usado para transmitir claves de distribución o datos de transacciones con los que contraer un acuerdo o contrato. Por ejemplo una clave DH de intercambio, una clave RSA para encriptación, etc. Su uso no está recomendado.

3.4.4. Elemento `<Object>`

Por último, el elemento opcional `<Object>` que puede aparecer una o más veces se utiliza para incluir objetos arbitrarios dentro de `<Signature>`, que pueden o no ser firmados.

Puede incluir los atributos *`Id`*, *`MimeType`* y *`Encoding`* de forma opcional.

3.5. Codificación base64

La codificación en el formato base64 documentada en la norma RFC 1421 permite transformar un fichero binario (en nuestro caso el resultado del algoritmo de firma) en texto en el que todos los bytes que se representan son caracteres imprimibles, de tal forma que se puede incluir como texto en un mensaje (en nuestro caso un documento XML).

Cuando se usa base64, se representan tres bytes binarios (24 bits en total) como cuatro caracteres imprimibles (cada uno 6 de bits).

Así cada grupo de 6 bits representa un valor entero de 0 a 63 al que se le asigna un carácter (debe ser imprimible) según la tabla 1 de la siguiente página.

Cuando se llega al final de un fichero es posible que sólo se tengan que convertir uno o dos bytes. En estos casos, se utiliza el carácter relleno '=' para completar la operación.

Si lo que queda por codificar al final de un fichero son dos bytes (16 bits), se suman dos bits iguales a cero para formar una secuencia de 18 bits que será la que se use para seleccionar tres caracteres de la tabla anterior (cada uno 6 bits).

Por último se añade un carácter de relleno '=' que hace de cuarto carácter. De esta forma se completan los 24 bits.

Si lo que queda por codificar al final de un fichero es un byte (8 bits), se suman cuatro bits iguales a cero para formar una secuencia de 12 bits que será la que se use para seleccionar dos caracteres de la tabla anterior (cada uno 6 bits). Por último se añaden dos caracteres de relleno '=' que hacen de tercer y cuarto carácter. De esta forma se completan los 24 bits.

Valor	Código	Valor	Código	Valor	Código	Valor	Código
0	A	16	Q	32	g	48	w
1	B	17	R	33	h	49	x
2	C	18	S	34	i	50	y
3	D	19	T	35	j	51	z
4	E	20	U	36	k	52	0
5	F	21	V	37	l	53	1
6	G	22	W	38	m	54	2
7	H	23	X	39	n	55	3
8	I	24	Y	40	o	56	4
9	J	25	Z	41	p	57	5
10	K	26	a	42	q	58	6
11	L	27	b	43	r	59	7
12	M	28	c	44	s	60	8
13	N	29	d	45	t	61	9
14	O	30	e	46	u	62	+
15	P	31	f	47	v	63	/

Tabla 1: Asignación caracteres imprimibles en la codificación base64

Además de todo esto la RFC 1421 especifica que los datos deben estar organizados en líneas de 64 caracteres, y que la última línea puede contener menos de 64 caracteres.

3.6. Identificadores de *Algorithm*

Digest

1. Required SHA1
http://www.w3.org/2000/09/XMLDSig#sha1

Encoding

1. Required base64
http://www.w3.org/2000/09/XMLDSig#base64

MAC

1. Required HMAC-SHA1
http://www.w3.org/2000/09/XMLDSig#hmac-sha1

Signature

1. Required DSAwithSHA1 (DSS)
http://www.w3.org/2000/09/XMLDSig#dsa-sha1

2. Recommended RSAwithSHA1
http://www.w3.org/2000/09/XMLDSig#rsa-sha1

Canonicalization

1. Required Canonical XML (omits comments)
http://www.w3.org/TR/2001/REC-xml-c14n-20010315

2. Recommended Canonical XML with Comments
http://www.w3.org/TR/2001/REC-xml-c14n-20010315#WithComments

Transform

1. Optional XSLT
http://www.w3.org/TR/1999/REC-xslt-19991116

2. Recommended XPath
http://www.w3.org/TR/1999/REC-xpath-19991116

3. Required Enveloped Signature
http://www.w3.org/2000/09/XMLDSig#enveloped-signature

3.7. Proceso de creación y validación de firmas en XML

En esta sección se detallan los pasos necesarios para crear y validar las firmas digitales que siguen la recomendación XML-Signature Syntax and Processing elaborada por W3C.

3.7.1. Creación de una firma digital

i. Determinar qué objetos deben firmarse e identificarlos mediante un identificador *URI*. Por lo tanto debemos crear una *URI* que identifique que firmaremos el objeto en el que está contenido el elemento `<Signature>` (es decir, el mensaje XML).

ii. Calcular la canonización y el *hash* de cada una de estas referencias y construir el elemento `<Reference>` para cada una de ellas. Creamos un elemento `<Reference>` que contenga su atributo URI. Creamos el elemento `<Transforms>` indicando la transformación Enveloped Signature y canonización de ese objeto e indicamos el *hash* aplicado en `<DigestMethod>`. El resultado de aplicar las transformaciones, canonizar y realizar el *hash* lo ponemos en la etiqueta `<DigestValue>`.

iii. Crear el elemento `<SignedInfo>`. Crear este elemento mediante la creación de los elementos `<Reference>`, `<SignatureMethod>` y `<CanonicalizationMethod>`.

iv. Canonizar y calcular la firma (e incluirla en `<DigestValue>`) sobre el elemento `<SignedInfo>` basándose en los algoritmos especificados en `<SignedInfo>`.

Primero calcular la canonización del elemento `<SignedInfo>`. Una vez obtenida la forma canónica de este elemento, calcular el *hash* y la firma sobre la forma canónica. El resultado se incluye en el elemento `<SignatureValue>`. Crear el atributo *Id* en el elemento `<SignatureValue>` de forma que sea único.

v. Generar la información de claves identificándose en el elemento `<KeyInfo>`. Incluir también la referencia al certificado asociado.

vi. Construir el elemento `<Signature>` que incluye los elementos `<SignedInfo>`, `<SignatureValue>` y `<KeyInfo>`.

3.7.2. Proceso de validación de una firma digital

i. Validación de la firma. Validaremos el elemento `<SignedInfo>`. Utilizar el elemento `<KeyInfo>` para recuperar la clave y la referencia al certificado asociado. Obtener la forma canónica del elemento `<SignedInfo>` utilizando el algoritmo descrito en el elemento `<CanonicalizationMethod>`. Recalcular el *hash* de esta forma canónica del elemento `<SignedInfo>` utilizando el algoritmo de *hash* especificado por el elemento `<SignatureMethod>`. Utilizar la clave para verificar que el valor de la firma contenido en el elemento `<SignatureValue>` es correcto y coincide con el resultado del *hash* aplicado a la forma canónica del elemento `<SignedInfo>`.

ii. Validación de las referencias. Validaremos que el *hash* contenido en los elementos `<DigestValue>` de cada elemento `<Reference>` es correcto.

    Para realizar esta validación es importante que antes del *hash* se realicen las dos transformaciones especificadas:

    - La transformación Enveloped Signature que elimina el elemento `<Signature>` del documento antes de canonizarlo.

    - Canonizar el documento sin el elemento de firma antes de realizar el *hash* mediante el algoritmo especificado en el elemento `<DigestMethod>`, y comprobar que coincide con el valor incluido en el elemento `<DigestValue>`.

Si ambos pasos dan un resultado positivo, entonces la firma es válida y correcta.

3.8. Implementaciones de la especificación XMLDSig

La implementación de la especificación de la firma digital XML (XMLDSig) elaborada por el W3C es posible llevarla a la práctica para muchas plataformas y lenguajes, aunque sin duda el lenguaje más utilizado es Java porque proporciona códigos independientes de la plataforma.

Además la elección de XML no es casual, ya que nos proporciona datos independientes de la plataforma, y si quisiéramos obtenerlos en otros formatos, como por ejemplo HTML u otros lenguajes de marcado, las transformaciones XSL (XSLT) que forman parte de la hoja de estilo XML estándar resuelven el problema de separar la presentación de los datos.

De esta forma, con un lenguaje independiente del sistema y un modelo de datos independiente obtenemos un programa robusto y portable junto con información fácil de traducir a otros formatos.

En el caso concreto de Java, ya existe un conjunto de librerías estándar para la firma digital XML de manera que se pueda utilizar en el Java 2 SDK por defecto, la conocida JSR 105 XML *Digital Signature API*.

En la página web de *Java Community Process* podemos encontrar la especificación para descarga en el apartado *Java Specification Request* bajo el estatus de *Final Release* y que tiene como objetivo proporcionar los servicios de la firma digital XML a través de un conjunto de API independientes.

Los miembros y colaboradores oficiales de la JSR 105 pertenecen en su mayoría a conocidos organismos independientes y a empresas del sector (SUN Microsystems lideró el proyecto), lo que nos hace pensar que ellas también han implementado sus propias clases para soportar el estándar XMLDSig. De esta forma aparecieron numerosos proveedores que entregan gratuitamente kits de desarrollo (SDK) con la documentación necesaria para utilizar los servicios de su API en particular, enriqueciendo el conjunto de herramientas que los programadores pueden seleccionar según su entorno de trabajo y que no les ata a lo que ocurra en la JSR promovida por SUN Microsystems.

A continuación mostramos algunas de las empresas y organismos que han creado sus propias implementaciones:

- KeyTools XML, Baltimore
- Apache XML Security for Java, Christian Geuer-Pollmann
- Entrust/Toolkit™; for Java™
- XML Signature Library (IXSIL), IAIK
- XML Security Suite, IBM
- XML Security Library, Aleksey Sanin
- VeriSign
- Infomosaic
- Microsoft
- NEC XMLDSIG

3.9. Conclusiones

XML es una tecnología sencilla que tiene a su alrededor otras que la complementan ofreciéndoles unas posibilidades mucho mayores. Tiene un papel muy importante en la actualidad ya que permite la compatibilidad entre sistemas para compartir la información de una manera segura, fiable y fácil. No es de extrañar que muchas de las especificaciones orientadas a seguridad hagan uso de este lenguaje.

Como hemos visto, la utilidad de XML es tal que no sólo se usa para definir la sintaxis de otros lenguajes, sino que se propone como un estándar para el intercambio de información estructurada entre diferentes plataformas. Se puede usar en bases de datos, editores de texto, hojas de cálculo y casi cualquier cosa imaginable. Es por eso que forma parte inherente de multitud de recomendaciones, como es el caso estudiado de XMLDSig.