XML SIGNATURE E XML ENCRYPTION

1 INTRODUZIONE

Il linguaggio XML (eXstensible Markup Language) ha acquistato, negli ultimi anni, un'importanza sempre crescente come standard fondamentale per la rappresentazione dei dati e per lo scambio di informazioni tra applicazioni che si trovano a cooperare attraverso il web. Ciò ha, di conseguenza, creato la necessità di modelli e meccanismi per la protezione di documenti XML.

Lo sviluppo di tecniche crittografiche orientate ad XML pone diversi nuovi requisiti rispetto alle tradizionali tecnologie che forniscono funzionalità di sicurezza.

Per risolvere questi problemi, il W3C, l'IETF, ed altre organizzazioni hanno lavorato per stabilire delle specifiche standard nell'ambito di quella che viene definita *XML security*. In particolare, le specifiche *XML Signature* e *XML Encryption* definite dal W3C rappresentano la base di tutti gli standard che concernono la sicurezza in XML.

Questo lavoro illustra queste due specifiche e le tecnologie correlate. In particolare nel prosieguo di questo capitolo introduttivo verranno discusse le motivazioni che hanno reso necessario lo sviluppo di tecniche di sicurezza specifiche per dati XML. Nella capitolo 2 verranno discusse le tecniche per la firma digitale definite dalla famiglia di standard XML Signature. Nella capitolo 3, invece, verrà trattato lo standard XML Encryption che definisce una sintassi XML per la rappresentazione di risorse web cifrate.

1.1 Le motivazioni per gli standard XML Signature e XML Encryption

A prima vista, la cifratura (o la firma digitale) di un documento XML potrebbe sembrare non differente rispetto alla cifratura di un documento qualsiasi.

Infatti, poiché un documento XML è essenzialmente un flusso di caratteri, può essere cifrato o firmato nella stessa maniera di qualsiasi altro tipo di documento. Qualunque applicazione crittografica che può cifrare o firmare un file generico, può benissimo essere usata per un file XML. Il problema è che, in questo modo, si pregiudica l'utilizzo delle caratteristiche vantaggiose della sintassi XML.

In particolare, tra le difficoltà che si possono incontrare con l'utilizzo di metodi crittografici tradizionali ci sono:

- *L'impossibilità delle normali firme digitali di gestire le modifiche causate dal parsing e dalla serializzazione*.

  E' abbastanza comune, per le applicazioni basate su XML, eseguire il parsing dei documenti in memoria (usando DOM o altre rappresentazioni) e, quindi, serializzarli in un file per l'immagazzinamento o la trasmissione. Questo processo può introdurre delle lievi modifiche nella rappresentazione del documento a livello di byte. Tali modifiche, sebbene non facciano differenza al fine del significato del documento, invalidano qualsiasi firma digitale che su quel documento eventualmente è stata precedentemente calcolata.

- *L'impossibilità delle firme digitali tradizionali di gestire i cambiamenti del set di caratteri*.

  I documenti XML possono essere rappresentati (codificati) con un arbitrario set di caratteri. Ad esempio, il carattere Ü (U con un umlaut) nella codifica ISO-8859-1 è rappresentato in forma esadecimale come 0xDC, mentre nella codifica UTF-8 è rappresentato da due byte, 0xC3 e 0x9C. Di conseguenza, due versioni dello stesso documento che usano codifiche differenti avranno due firme digitali diverse, anche se il significato del documento è lo stesso.

- *L'impossibilità di rappresentare il valore di una firma digitale o l'output di un documento XML cifrato in formato XML*.

  L'output dei tradizionali meccanismi di cifratura e firma digitale è costituito da dati binari. Un requisito fondamentale delle applicazioni basate su XML è, invece, quello di poter specificare l'output delle operazioni crittografiche in formato XML in modo che i documenti risultanti possano essere processati semplicemente con i comuni e diffusi tools XML. In questo modo i meccanismi di sicurezza possono essere integrati nell'infrastruttura applicativa in maniera efficiente assicurando, inoltre, l'interoperabilità tipica di XML.

Gli standard *XML Signature* e *XML Encryption* del World Wide Web Consortium (W3C) sono stati sviluppati allo scopo di superare queste difficoltà, facendo uso delle caratteristiche di XML per sviluppare un framework flessibile per le funzionalità crittografiche di cifratura e firma digitale. Fondamentalmente, sono basati sui concetti, gli algoritmi e le tecnologie dei sistemi di sicurezza tradizionali, con l'aggiunta delle necessarie funzionalità che permettono di supportare l'integrazione con XML.

2 XML SIGNATURE

2.1 Panoramica

Lo standard XML Signature fornisce un insieme di tecniche per il calcolo della firma digitale specifiche per l'utilizzo in transazioni XML. Esso definisce uno schema per la rappresentazione in formato XML del risultato di un'operazione di firma digitale applicata a dati arbitrari. Così come gli schemi di firma digitale tradizionali, XML Signature fornisce le funzionalità di autenticazione, integrità e supporto alla non ripudiabilità sui dati che vengono firmati. Tuttavia, rispetto agli standard tradizionali, XML Signature è stato progettato tenendo in considerazione le caratteristiche

peculiari del linguaggio XML e le modalità con cui i documenti XML vengono distribuiti e recuperati sulla rete Internet.

Una caratteristica fondamentale di XML Signature è la capacità di poter firmare solo specifiche porzioni di un documento XML. Ciò è particolarmente importante in scenari dove un documento XML è costituito da componenti rilasciate da entità differenti, ognuna delle quali può specificare una firma per la sola parte che le compete. Tale flessibilità è altrettanto rilevante in situazioni dove è importante assicurare l'integrità di determinate porzioni di un documento XML, lasciando invece la possibilità di poter modificare altre parti.

Un'altra proprietà di XML Signature è quella che consente ad una singola firma di potersi riferire contemporaneamente a diversi tipi di risorsa come, ad esempio, dati testuali (HTML), dati binari (JPEG), o specifiche sezioni di documenti XML.

Il documento che specifica una firma XML contiene l'indicazione sulla locazione degli oggetti di dati firmati che può essere espressa per mezzo di diversi meccanismi. In particolare, i dati a cui è stata apposta una firma possono:

- essere esterni al documento XML che contiene la firma e localizzati attraverso un riferimento URI;
- risiedere nello stesso documento XML che contiene la firma (la firma viene inclusa nel documento come elemento fratello di quello che contiene i dati);
- essere inclusi nella firma (come elemento figlio);
- contenere al loro interno la firma (come elemento figlio).

2.2 XML Signature e specifiche correlate

Le specifiche XML Signature sono frutto del lavoro congiunto del World Wide Web Consortium (W3C) e dell'Internet Engineering Task Force (IETF) il cui scopo è stato quello di sviluppare una sintassi conforme al linguaggio XML per la rappresentazione di firme digitali di risorse web (in particolare documenti XML) e la definizione di procedure per il calcolo e la verifica di tali firme.

XML Signature è stata rilasciata come Raccomandazione ufficiale del W3C nel febbraio del 2002. In aggiunta, il W3C ha rilasciato delle specifiche correlate che forniscono funzionalità essenziali nell'uso di XML Signature.

L'insieme attuale delle specifiche nell'ambito XML Signature sono le seguenti:

- XML-Signature Syntax and Processing, Second Edition (Raccomandazione W3C, 2008).
- Canonical XML Version 1.0 (Raccomandazione W3C, 2001).
- Exclusive XML Canonicalization Version 1.0 (Raccomandazione W3C, 2002).
- XML-Signature XPath Filter 2.0 (Raccomandazione W3C, 2002).

2.3 XML-Signature Syntax and Processing

La specifica *XML-Signature Syntax and Processing* costituisce il nucleo fondamentale della famiglia di standard XML Signature. Essa definisce la sintassi XML per la rappresentazione di una firma digitale e le regole per la creazione e la verifica di tali firme.

Una firma digitale XML Signature consiste in un documento XML che contiene tutte le informazioni sui dati che vengono firmati e sul valore della firma, oltre quelle necessarie al processo di verifica.

È possibile firmare contemporaneamente diverse risorse, che possono essere locali al documento che contiene la firma oppure remote ed accessibili tramite un Uniform Resource Identifier (URI). Tali risorse sono espresse usando XML, il quale permette delle differenze sintattiche su documenti logicamente equivalenti. Ciò significa che è possibile per due documenti XML essere semanticamente equivalenti nonostante delle differenze a livello di byte (un semplice esempio è l'aggiunta di un carattere spazio all'interno di un tag). Questa libertà nel formato causa dei problemi alle funzioni hash (che sono alla base delle operazioni di firma digitale), le quali operano a livello di byte. Per risolvere questo problema vengono usati degli algoritmi di canonicalizzazione che permettono di rimuovere le differenze sintattiche tra documenti semanticamente equivalenti. Tali algoritmi assicurano che la funzione hash sia applicata allo stesso insieme di byte.

Le funzioni hash vengono usate in due momenti differenti nella generazione di una firma XML: dapprima viene calcolato il valore hash di ogni singola risorsa che partecipa alla firma e, quindi, sulla lista di risorse viene calcolato l'hash una seconda volta durante l'operazione di firma.

La figura seguente mostra la struttura di una firma digitale XML.

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

Figura 1

Gli operatori di cardinalità (le cui definizioni sono date nella tabella seguente) denotano le occorrenze di ogni elemento all'interno dell'elemento contenitore `<Signature>`.

OPERATORE	DESCRIZIONE
*	Zero o più occorrenze
+	Una o più occorrenze
?	Zero o una occorrenza

Tabella 1

La figura seguente mostra una semplice istanza di una firma XML con la quale verrà introdotto il significato e la funzione dei principali elementi della sintassi.

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

```
    <Reference URI="file:///C:\check.txt">
      <DigestMethod Algorithm="http://www.w3.org/2000/09/
                               xmldsig#sha1"/>
      <DigestValue>
        aZh8Eo2alIke1D5NNW+q3iHrRPQ=
      </DigestValue>
    </Reference>
  </SignedInfo>
  <SignatureValue>
    MI6rfG4XwPFASDFfgAFsdAdASfasdFBVWxMrO/
    Ta7nm5SfQ26KxK71Ch+4wHCMyxEkBxx2HP0/7J
    tPiZTHNYTEWFtgWRvfwrfbvRFWvRWVnmi3ksdT
    Q+zMtKjQAsdfJHyheAWErHtw3qweavfwtRHyK1
    9ExbdFWQAEDafsf/f8I=
  </SignatureValue>
</Signature>
```

Figura 2

L'elemento `<Signature>` funge da contenitore per tutte le informazioni sulla firma. La lista delle risorse che devono essere firmate è contenuta nell'elemento `<SignedInfo>`. In particolare per ogni risorsa è presente un elemento `<Reference>` con un attributo URI che la identifica (nell'esempio è presente una sola risorsa specificata dall'URI `file:///C:\check.txt`). Inoltre, ogni elemento `<Reference>` ha due elementi figli che specificano un valore digest della risorsa e l'algoritmo con il quale tale valore viene calcolato (nell'esempio SHA-1). In XML Signature, infatti, non vengono firmate direttamente le risorse, ma i valori hash calcolati sulle stesse. Ciò, oltre ad aumentare l'efficienza dell'operazione di firma, fornice un metodo per la firma di risorse multiple. Nell'elemento `<SignedInfo>` è presente anche un elemento `<SignatureMethod>` che contiene un attributo Algorithm che specifica il tipo di algoritmo di firma usato (nell'esempio, RSA con SHA-1).

Il valore della firma consiste nel risultato dell'applicazione dell'algoritmo di firma e della funzione hash scelte sul contenuto dell'elemento `<SignedInfo>`. Tale valore viene salvato nell'elemento `<SignatureValue>` codificato in Base-64.

La codifica Base-64 è molto usata nelle applicazioni basate su XML in quanto costituisce un meccanismo robusto e conveniente per ottenere una rappresentazione testuale di dati binari arbitrari.

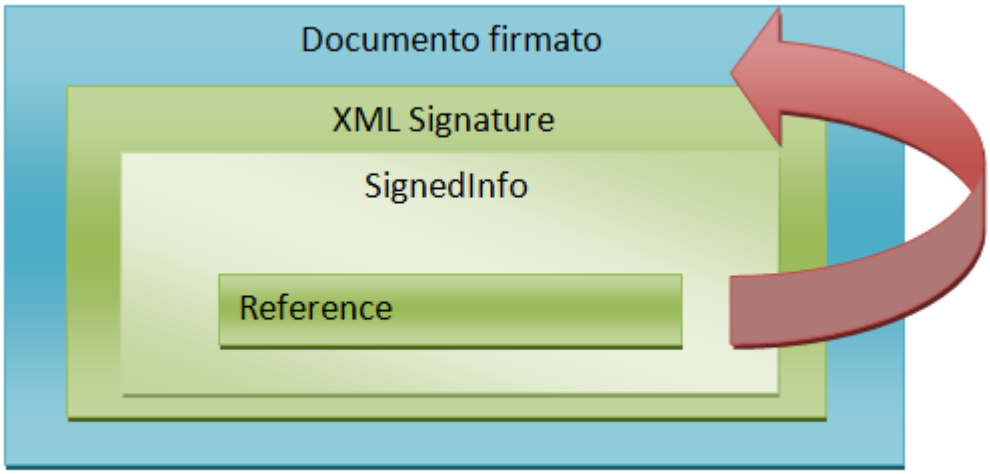
Nel resto del paragrafo verranno dapprima discusse le varie tipologie di firma XML Signature, per poi addentrarsi nella descrizione degli elementi che ne fanno parte. Infine saranno illustrate le procedure per creare e validare una firma XML.

2.3.1 Tipi di firma XML

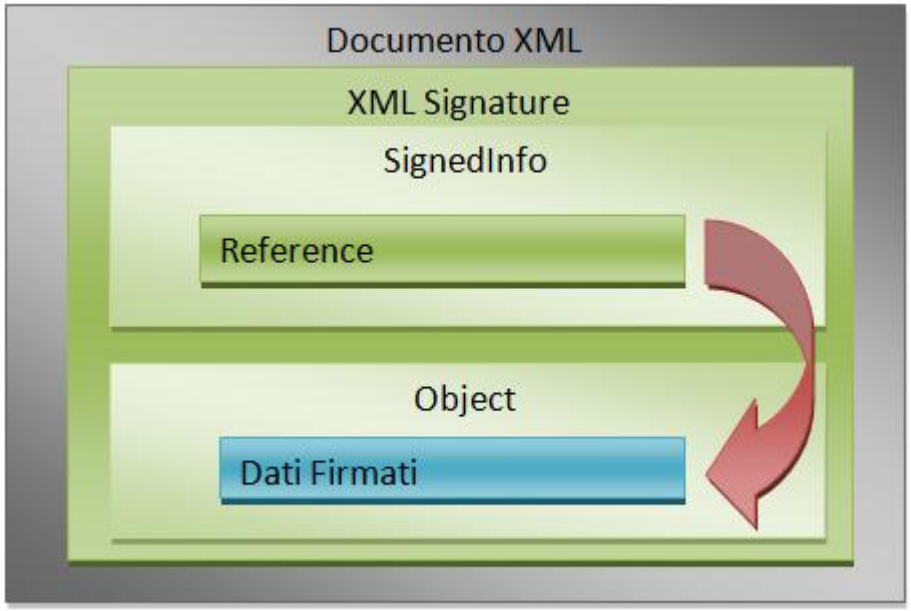
La specifica XML Signature prevede tre tipi di formato per la rappresentazione di una firma digitale, che si differenziano per la localizzazione dei dati che vengono firmati. Queste differenze vengono incontro ai diversi requisiti delle varie applicazioni. Alcune di esse, ad esempio, possono richiedere che una firma XML sia quanto più possibile conforme ad una firma reale, cioè che sia inclusa in qualche parte del documento originale. Altre applicazioni, invece, possono processare i dati originali separatamente dalla firma e possono richiedere che essi siano rimossi dalla firma stessa.

Se il documento originale è 'strettamente connesso' all'elemento `<Signature>` che contiene la firma, allora si avrà una firma di tipo *enveloped* (se il documento originale è padre dell'elemento `<Signature>`) oppure di tipo *enveloping* (se il documento originale è un figlio dell'elemento `<Signature>`). Quando invece il documento originale non ha relazioni di padre-figlio con l'elemento `<Signature>`, allora la firma è di tipo *detached*.

La figura seguente mostra visivamente la relazione con i dati di ognuno dei tipi di firma appena elencati.



ENVELOPING SIGNATURE



DETACHED SIGNATURE

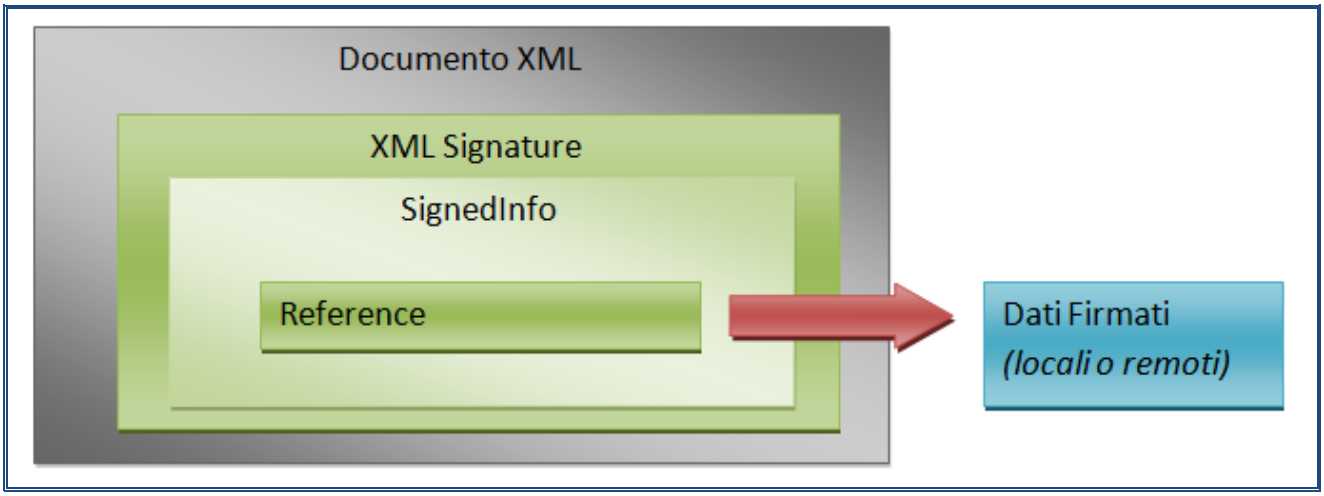


Figura 3

Come si può notare dalla figura, una firma di tipo *enveloped* deve essere figlio dei dati da firmare, mentre una firma di tipo *enveloping* deve essere padre dei dati da firmare. Infine una firma di tipo *detached* non ha rapporti di padre-figlio con i dati da firmare, i quali possono essere locali al documento che contiene la firma oppure remoti.

Una singola firma può anche essere una combinazione di più di uno dei tipi descritti. Infatti, è possibile per un elemento `<Signature>` avere elementi `<Reference>` multipli, ognuno dei quali può puntare sia a dati locali al blocco `<Signature>` che a dati localizzati in remoto.

La figura seguente mostra un esempio di un blocco `<Signature>` che è allo stesso tempo di tipo enveloping e detached.

```
<Signature>
  <SignedInfo>
    <SignatureMethod
      Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
    <Reference URI="http://www.myserver.com/importantFile.xml">
      <DigestMethod
Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
      <DigestValue>aZh8Eo2alIke1D5NNW+q3iHrRPQ=</DigestValue>
      </Reference>
    <Reference URI="#ImportantMessage">
      <DigestMethod
Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
      <DigestValue>qGh8Eo2alJke1D7NNW+z3iHhRPF=</DigestValue>
```

```
    </Reference>
  </SignedInfo>
  <SignatureValue>
    MI6rfG4dwPFASDFfgAFsdAdASfasdFBVWxMrO/
    Ta7nFCSDAhnTRhy45vJSDcvadrtrEQW2HP0/7J
    tPQTBFfwGVwfqewrfVfewgrtgvfbwdj7jujYdT
    Q+zMtKRQGRE1grewrfht32rwhnbtygrwtRHyK1
    3EFfdasreEDafsfgf8I=
  </SignatureValue>
  <Object>
    <original_document>
      <very_important_element id="ImportantMessage">
        Milk Chocolate is better than Dark Chocolate!
      </very_important_element>
    </original_document>
  </Object>
</Signature>
```

Figura 4

Qui, i due elementi <Reference> mostrati in rosso, puntano a dati localizzati in posti differenti. Infatti il primo si riferisce all'elemento <original_document> interno al documento, mentre il secondo punta a dati esterni al documento (il file importantFile.xml). Si tratta quindi di una firma che ha la doppia proprietà: enveloping e detached.

2.3.2 Sintassi di XML Signature

Di seguito si daranno delle descrizioni degli elementi principali della sintassi XML Signature, cercando per ognuno di descriverne la natura e l'intento.

Elemento <Signature>

L'elemento radice di una firma XML Signature è <Signature>. Tale elemento rappresenta il contenitore di una firma completa in un dato contesto. Può avere come elementi figli <SignedInfo>, <SignatureValue>, <KeyInfo> e <Object> (gli ultimi due sono opzionali). L'elemento <Signature> può

contenere un attributo `Id` come identificatore, utile nel caso di istanze multiple di elementi `<Signature>` in un singolo file o contesto.

Elemento <SignedInfo>

L'elemento `<SignedInfo>` è, tra tutti, quello più complesso (ha più figli). Come indica il nome, tutte le informazioni in esso contenute vengono firmate, in particolare contiene un riferimento ad ogni oggetto di dati che deve essere incluso nella firma. È costituito da una sequenza dei seguenti elementi: `<CanonicalizationMethod>`, `<SignatureMethod>` e uno o più elementi `<Reference>`.

L'elemento `<CanonicalizationMethod>` indica l'algoritmo usato per generare la forma canonica dell'elemento `<SignedInfo>`.

L'elemento `<SignatureMethod>` identifica l'algoritmo usato per produrre il valore della firma contenuto nell'elemento `<SignatureValue>`.

Questi due elementi contengono semplicemente degli identificatori. Essi vengono inclusi come componenti di `<SignedInfo>` al fine di prevenire attacchi di sostituzione. Ad esempio, se l'elemento `<SignatureMethod>` venisse esplicitamente definito all'infuori del blocco `<SignedInfo>`, allora un attaccante potrebbe modificare l'identificatore che descrive il metodo usato per la firma, provocando il fallimento di qualsiasi processo di validazione.

L'elemento `<Reference>` referenzia attraverso un URI (un meccanismo universale per riferirsi a dati locali o remoti) un flusso di dati di cui sarà calcolato un valore hash. Tale elemento verrà trattato in dettaglio nel seguito.

La figura seguente mostra una firma XML in cui vengono evidenziati gli elementi che costituiscono `<SignedInfo>`.

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

```
    <SignatureMethod Algorithm="http://www.w3.org/
        2000/09/xmldsig#rsa-sha1"/>
    <Reference URI="http://www.rsasecurity.com">
    <DigestMethod Algorithm="http://www.w3.org/
      2000/07/xmldsig#sha1"/>
      <DigestValue>aZh8Eo2alIke1D5NNW+q3iHrRPQ=</DigestValue>
    </Reference>
  </SignedInfo>
  ...
</Signature>
```

Figura 5

Nell'esempio considerato, l'elemento `<CanonicalizationMethod>` specifica un metodo di canonicalizzazione richiesto dalla specifica XML Signature. Si tratta del metodo '*Canonical XML Without Comments*' definito dall'URI `http://www.w3.org/TR/2001/REC-xml-c14n-20010315`. In questo caso l'URI è semplicemente un identificatore, a differenza di quella specificata nell'elemento `<Reference>` (`http://www.rsasecurity.com`) che rappresenta invece un flusso di dati (di cui si calcola un digest che viene successivamente firmato).

L'elemento che segue `<SignedInfo>`, all'interno di `<Signature>`, è `<SignatureValue>` che consiste essenzialmente in un contenitore per il valore codificato della firma.

Elemento <Reference>

L'elemento `<Reference>` costituisce il fulcro di XML Signature. Esso descrive come le risorse vengono recuperate ed eventualmente trasformate per produrre i dati su cui viene calcolato un digest che successivamente viene firmato come parte dell'elemento `<SignedInfo>`. Un elemento `<Reference>` può rappresentare una risorsa di qualsiasi tipo di dati. Questo punto è particolarmente importante in quanto spesso si assume erroneamente che XML Signature possa essere usato per firmare esclusivamente dati XML.

Molte delle caratteristiche fondamentali di XML Signature (come la firma selettiva) sono applicabili quando la sorgente di dati da firmare è un documento XML. Tuttavia

ciò non preclude che byte di dati arbitrari possano essere ugualmente firmati con le tecniche di XML Signature. Anche un documento XML, se referenziato come risorsa remota, è trattato come un file generico e processato come un flusso di byte. I documenti XML sono processati in maniera specifica solo nel caso di firma di tipo *enveloped* e *enveloping*.

L'elemento `<Reference>` contiene tre elementi figli: `<Transforms>`, `<DigestValue>` e `<DigestMethod>`, e tre attributi opzionali: `Id`, `Type` e `URI`.

L'attributo `Id` è un identificatore generico che può essere usato in caso di riferimenti remoti.

L'attributo `Type` specifica il tipo di dati puntato da `<Reference>`. Il valore di tale attributo è da intendersi in senso generico, piuttosto che come descrizione del formato dei dati specificati. Consideriamo, ad esempio, il seguente elemento `<Reference>`:

```
<Reference Type="http://www.w3.org/2000/09/xmldsig#Object"
           URI="#myobject">
  <DigestMethod Algorithm="http://www.w3.org/2000/09
                /xmldsig#sha1" />
  <DigestValue>
    70NvZxcdTB+7UnlLp/J724p8h4zx=
  </DigestValue>
</Reference>
```

Supponiamo, quindi, che il riferimento corrisponda ad un elemento `<Object>` identificato dall'attributo `myobject` e contenente dati codificati in Base-64. Sia esso:

```
<Object Id="myobject">
  lczCCAbYwggErBgcqhkjOOAQBMIIBHgKBgQDaJjfDTrawMHf8MiUt
  Y54b37hSmYNnR3KpGT10uU1Dqppcju06uN0iGbqf947DjkBC25hKnq
</Object>
```

L'esempio mostra come l'attributo `Type` di `<Reference>` non si riferisce ai dati, ma al contenitore in cui questi risiedono (in questo caso un elemento `<Object>`).

L'attributo più importante dell'elemento `<Reference>` è URI. Esso costituisce un puntatore alla risorsa di dati su cui verrà calcolato il digest. Prima di questa operazione, i dati possono essere sottoposti a varie trasformazioni. In pratica, il flusso di byte su cui si calcola il digest, è il risultato di alcune trasformazioni che

vengono specificate da un elemento <Transforms>. Quest'ultimo è un contenitore per uno o più elementi <Transform> che specificano il tipo di trasformazione da eseguire.

Le trasformazioni definite dagli elementi <Transform> lavorano in modo seriale, cioè l'output di una diviene l'input della successiva rispettando l'ordine in cui appaiono come figli del contenitore <Transforms>.

La figura seguente usa l'analogia con una cascata per mostrare il flusso di dati che dalla sorgente (i dati puntati da un URI), procede attraverso le varie trasformazioni fino a diventare l'input della funzione di digest.

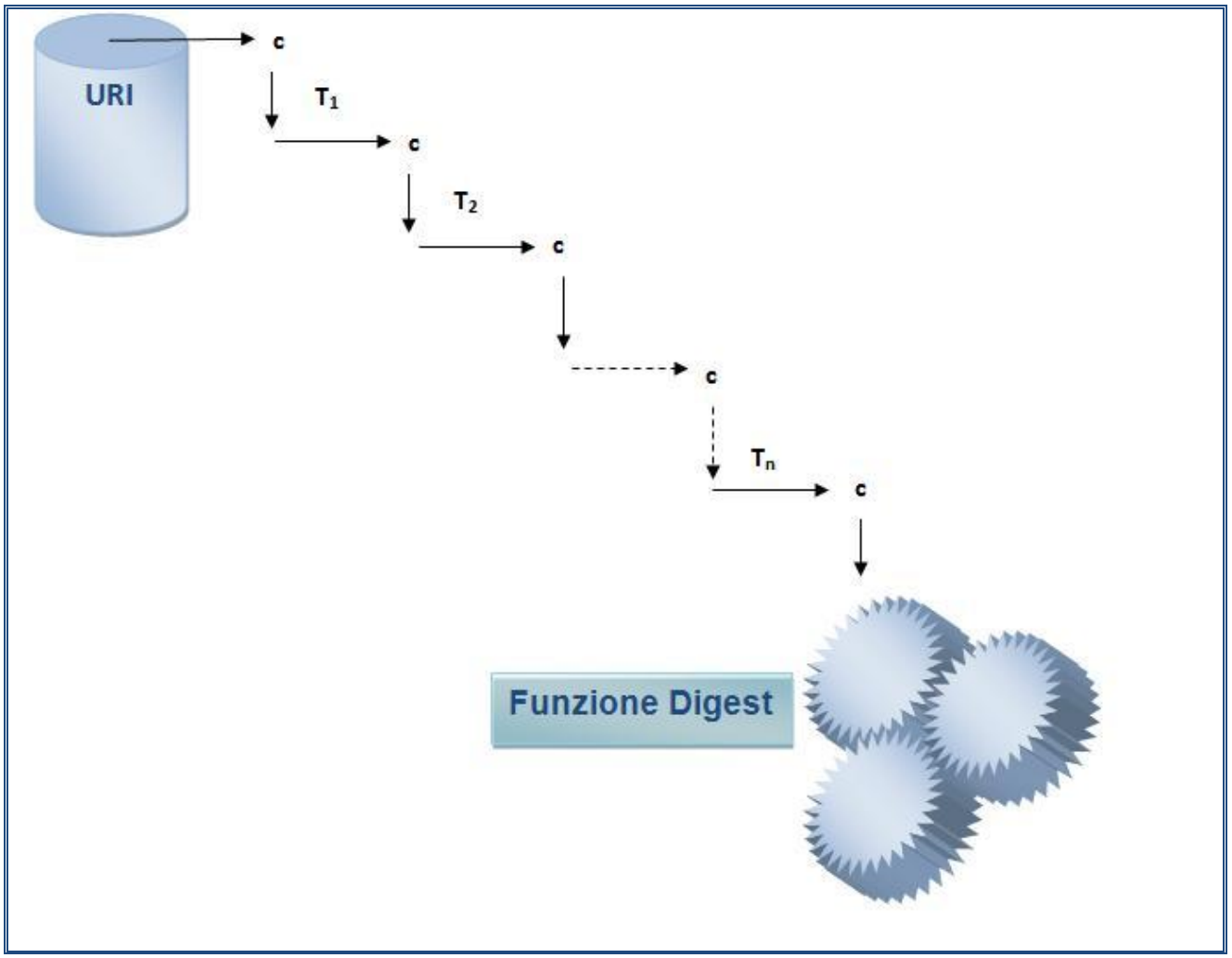


Figura 6 (T_1…T_n rappresentano le trasformazioni sui dati – La notazione "c" denota una possibile conversione da node-set a flusso di byte o viceversa)

In questa figura le frecce orizzontali rappresentano le trasformazioni effettuate sui dati, mentre quelle verticali indicano una possibile conversione tra flusso di byte e node-set o viceversa. Ciò perché alcuni algoritmi di trasformazione richiedono che i dati da processare siano sottoforma di node-set, mentre altri accettano esclusivamente flussi di byte binari.

Quindi, il ruolo di un insieme di trasformazioni è quello di rappresentare i dati che devono essere sottoposti al calcolo di un digest. Ad un livello più concreto, una singola trasformazione è semplicemente un algoritmo che opera su un input: il flusso di dati specificato da un URI o l'output di una precedente trasformazione.

La specifica XML Signature menziona cinque trasformazioni di default:

- Canonical XML;
- Base-64 decoding;
- XPath filtering;
- Enveloped transform;
- XSLT transform.

Tali trasformazioni operano su due tipi di dati: flusso di byte o node-set XPath. Nel secondo caso i dati vengono processati come XML.

Una volta applicate ai dati tutte le trasformazioni specificate, sul risultato viene calcolato un valore digest con il metodo specificato dall'attributo URI dell'elemento `<DigestMethod>`. Tale valore viene salvato nell'elemento `<DigestValue>`.

Di seguito viene mostrata un'istanza di un elemento `<Reference>` che contiene tutti i possibili attributi ed elementi figli.

```
<Reference Id="Full-Reference"
           URI_"#EnvelopedText"
           Type_"http://www.w3.org/2000/09/xmldsig#Object">
  <Transforms>
    <Transform Algorithm="http://
                          www.w3.org/2000/09/xmldsig#base64"/>
  </Transforms>
  <DigestMethod Algorithm="http://
                           www.w3.org/2000/09/xmldsig#sha1"/>
  <DigestValue>
    90NvBvtdTB_7UnlLp/V424p8o5cxs_
  </DigestValue>
</Reference>
```

Elemento <KeyInfo>

L'elemento `<KeyInfo>` (opzionale) permette l'integrazione di meccanismi di fidatezza nelle applicazioni che utilizzano le firme XML Signature. Esso contiene informazioni specifiche usate per la verifica di una firma. L'informazione può essere esplicita, come una chiave pubblica o un certificato X.509, oppure può essere indiretta e specificare un'entità remota.

La logica definita da `<KeyInfo>` permette che il problema della verifica della fidatezza sia di competenza del dominio applicativo che in questo modo ha più flessibilità nel decidere i propri meccanismi.

Un modo per gestire queste informazioni in un'applicazione che si basa su XML Signature è quello di delegare il compito ad un componente specializzato separato (*trust engine*) che prende in input l'elemento `<KeyInfo>` e decide in base al suo contenuto.

La figura seguente mostra come un documento XML che contiene un elemento `<Signature>` possa essere elaborato per recuperare e interpretare l'elemento `<KeyInfo>`.

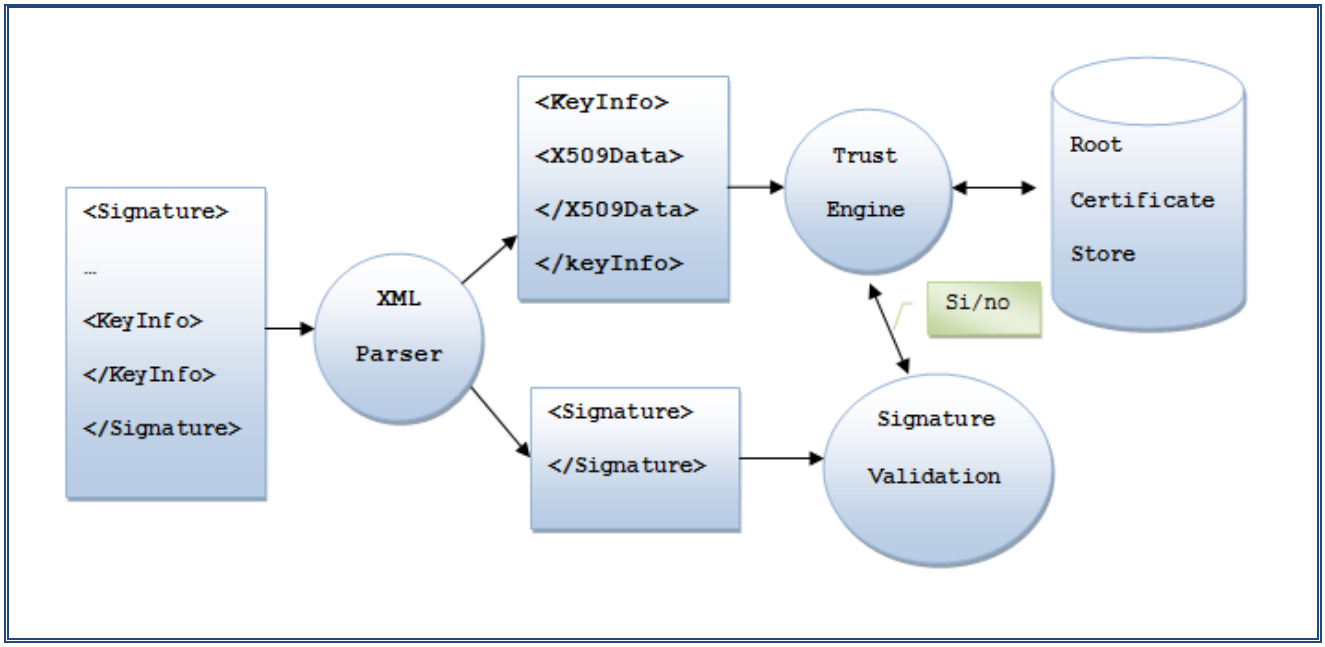


Figura 7

L'elemento <KeyInfo> nell'esempio contiene un certificato X.509 e viene passato al *trust engine* che comunica una decisione binaria (si/no) al componente per la validazione della firma. Si tratta dell'algoritmo *certificate path validation* definito dallo standard X.509 per il quale il certificato in <KeyInfo> viene cercato in uno store di *trusted root certificate*.

Ovviamene la certificate path validation non è l'unico metodo per asserire la fidatezza di materiale riguardante chiavi pubbliche. XML Signature permette l'utilizzo di diverse componenti all'interno di <KeyInfo>. La tabella seguente descrive i vari elementi che possono essere inclusi.

Elemento	Descrizione
<KeyName>	Un semplice identificatore testuale per il nome della chiave
<KeyValue>	Una chiave pubblica RSA o DSA
<RetrevialMethod>	Permette di riferirsi a informazioni sulla chiave localizzate in remoto
<X509Data>	Dati relativi a certificati X.509
<PGPData>	Identificatori e chiavi PGP
<SPKIData>	Dati, certificati e chiavi relativi a SPKI
<MgmtData>	Parametri per l'accordo sulla chiave (ad esempio parametri Diffie-Hellman)

Tabella 2

Quando all'interno di un singolo elemento <KeyInfo> compaiono diversi elementi, essi devono riferirsi alla stessa chiave. Questa restrizione serve per prevenire ambiguità nel processo di verifica della firma. La moltitudine di elementi disponibili per <KeyInfo> permette un alto grado di configurazione. Inoltre, è possibile per un'applicazione aggiungere elementi provenienti da altri namespace.

La natura estensibile di <KeyInfo> permette ad altre tecnologie XML di fornire propri meccanismi da integrare nella sintassi XML Signature. In particolare la specifica *XML Key Management Specification* definisce un protocollo per la distribuzione e la registrazione di chiavi pubbliche.

Elemento <Object>

L'elemento `<Object>` è un contenitore generico che può includere dati di ogni tipo. L'usuale utilizzo è quello di includere nell'elemento i dati che devono essere firmati dando vita ad una firma di tipo *enveloping*. La sola restrizione è che i dati binari inclusi in un elemento `<Object>` siano codificati in un formato conforme alla rappresentazione in un documento XML. Ciò di solito si realizza attraverso una codifica in Base-64, benché schemi di codifica alternativi siano comunque permessi da XML Signature.

L'elemento `<Object>` ha tre attributi opzionali: `Id`, `MimeType` e `Encoding`. L'attributo `Id` è usato per riferirsi all'elemento `<Object>` da un'altra posizione all'interno del blocco `<Signature>`. L'attributo `MimeType` serve ad indicare, ad un'applicazione che processa la firma, il tipo di dati contenuto in `<Object>`, indipendentemente da come questi sono codificati. L'attributo `Encoding` è un identificatore URI che descrive il tipo di meccanismo di codifica usato. La figura seguente mostra un esempio di inclusione di un file binario GIF che rappresenta un'immagine come parte di una firma di tipo *enveloping* con l'uso di un elemento `<Object>`.

```
<Signature>
  <SignedInfo>
    <Reference Type=http://www.w3.org/2000/09/xmldsig#Object
               URI="#ImportantPicture">
      <DigestMethod Algorithm="http://www.w3.org/
                    2000/09/xmldsig#sha1" />
    <DigestValue>HfRNHKuQrDiTy3XABMFbyteg3CG=</DigestValue>
    </Reference>
  </SignedInfo>
  <Object Id="ImportantPicture" MimeType="image/gif"
          Encoding="http://www.w3.org/2000/09/xmldsig#base64">
    aWcgQmxha2UncyBBdXRoZW50aWNhdGlvbiBTZXJ2aWNlMRQwEgY
    DVQQLEwtFbmdpbmVlcmluZzEWMBQGA1UEAxMNQmlnIEJhZCBCbG
    FrZTEcMBoGCSqGSIb3DQEJARYNYmJiQGJiYmFzLmNvbTAeFw0wM
    DA2MjAyMTEzMzVaFw0xMTA2MDMyMTEzMzVaMH4xCzAJBgNVBAYT
    AlVTMRMwEQYDVQQIEwpTb211LVN0YXRlMQ8wDQYDVQQKEwZTZXJ2
    ZXIxFDASBgNVBAsTC1NlcnZlciBDZXJ0MRMwEQYDVQQDEwpTZXJ2
    ZXJDZXJ0MR4wHAYJKoZIhvcNAQkBFg9zZXJ2ZXJAY2VydC5jb20Wg
```

```
    Z8wDQYJKoZIhvcNAQEBBQADY0AMIGJAoGBAMg7Y9ZByAKLTf4eOaN
    o8i5Ttge+fT1ipOpMB7kNip+qZR2XeaJCiS7VMetA5ysX7deDUYYk
    pefxJmhbL2hO+hXj72JCY0LGJEKK4eIf8LTR99LIrctz
  </Object>
  ...
</Signature>
```

Figura 8

Nell'esempio, l'elemento `<Reference>` usa l'attributo `Type` che identifica il tipo di oggetto puntato; in questo caso, un tipo `Object`. Tale attributo è opzionale e può essere omesso se l'applicazione può determinare il tipo attraverso altri meccanismi. L'attributo URI usato nell'elemento `<Reference>` è un meccanismo di puntamento a risorse XML che contengono un attributo `Id` con lo stesso identificativo; in questo caso, è l'elemento `<Object>` che ha "`ImportantPicture`" come valore dell'attributo `Id`. L'elemento `<Object>` mostrato usa tutti e tre gli attributi opzionali, in particolare `MimeType` specifica il valore "`image/gif`" che indica il tipo di dati.

2.3.3 XML Signature Processing

In questo paragrafo viene descritto il *processing model*, cioè le regole da seguire riguardo le operazioni che devono essere eseguite nella generazione e nella validazione di una firma XML da parte delle implementazioni dello standard.

La generazione di una firma è definita dalla Raccomandazione XML Signature come *core generation*, mentre la validazione *core validation*.

Core generation

La procedura *core generation* viene definita dallo standard in due fasi: *reference generation* e *signature generation*. Il primo definisce come viene calcolato il valore

digest di ogni elemento <Reference>, mentre il secondo specifica come viene calcolato il valore della firma contenuto in <SignatureValue>.

L'obiettivo della *reference generation* è di creare gli elementi <Reference> provvisti di tutte le componenti (quali trasformazioni, attributi e valori digest). Quello della *signature generation*, invece, consiste nel produrre il valore della firma e costruire l'intero blocco <Signature>.

La figura seguente mostra i passi delle *reference* e *signature generation*.

CORE GENERATION
Reference generation Per ogni risorsa da firmare: 1. Applica le trasformazioni (se presenti). 2. Calcola il valore digest delle risorse trasformate. 3. Crea un elemento <Reference> che includa gli attributi, le trasformazioni, gli identificatori per l'algoritmo di digest e il valore digest
Signature generation 1. Usando gli elementi <Reference> prodotti nella reference generation, crea un elemento <SignedInfo> che specifica un algoritmo di firma ed uno di canonicalizzazione. 2. Applica l'algoritmo di canonicalizzazione e quello di firma all'elemento <SignedInfo>. 3. Costruisci l'elemento <Signature> inserendo nell'elemento <SignatureValue> il valore della firma calcolato, e utilizzando l'elemento <SignedInfo> prodotto al passo precedente. Inserisci, inoltre, ulteriori elementi e attributi opzionali se richiesti.

Figura 9

È importante notare come, nel primo passo della signature generation, non si applicano ai dati gli algoritmi di canonicalizzazione e di firma, ma vengono semplicemente creati gli elementi con gli appropriati identificatori URI che descrivono tali algoritmi. Ciò perché i metodi usati per la firma e la

canonicalizzazione devono anch'essi essere firmati (e quindi inclusi nell'elemento <SignedInfo>) per resistere a possibili attacchi per sostituzione.

Nel secondo passo della signature generation ha luogo l'effettiva operazione di firma. L'oggetto di dati che viene firmato (<SignedInfo>), definisce gli algoritmi di firma e canonicalizzazione usati. In tutti i casi, la canonicalizzazione dell'elemento <SignedInfo>, prima che venga firmato, è obbligatoria. La ragione di ciò è data dal fatto che la rappresentazione fisica dell'elemento <SignedInfo> può cambiare quando il documento XML che lo contiene si sposta tra vari domini applicativi.

Core validation

Il processo di *core validation* descrive i passi per la validazione della firma e consiste nell'inverso della *core generation*. Anch'esso è composto da due fasi: *reference validation* e *signature validation*. Lo scopo della prima è verificare che i dati puntati dagli elementi <Reference> non siano stati alterati. Ciò avviene calcolando il digest della sorgente di dati comparando il valore ottenuto con quello salvato nell'elemento <DigestValue>. In maniera analoga, lo scopo della signature validation è confrontare il valore della firma calcolato sulla forma canonica di <SignedInfo> con quello salvato nell'elemento <SignatureValue>.

La serie di passi prevista dalla *core validation* è specificata nella figura seguente.

CORE VALIDATION
Reference validation Dapprima, l'elemento <SignedInfo> deve essere canonicalizzato. Per ogni elemento <Reference> che deve essere validato, esegui i seguenti passi: 1. Ottieni il flusso di dati su cui calcolare il digest attraverso l'attributo URI di <Reference>. Se non è presente tale attributo, l'applicazione deve conoscere la locazione della sorgente di dati. I dati finali sono il risultato delle trasformazioni applicate in cascata. 2. Sul flusso di dati ottenuto al passo 1 calcola il digest usando la funzione hash specificata nell'elemento <DigestMethod>.

3. Confronta il valore digest calcolato al passo 2 con quello contenuto nell'elemento `<DigestValue>`. Se i due valori non coincidono, la reference validation fallisce.

Signature Validation

1. Recupera la chiave per la verifica dall'elemento `<KeyInfo>` o da un'altra locazione nota all'applicazione.
2. Usando la forma canonica di `<SignatureMethod>`, determina l'algoritmo di firma e usalo per il calcolo della stessa sull'elemento `<SignedInfo>`. Confronta il valore ottenuto con quello contenuto nell'elemento `<SignatureValue>`. Se tali valori non coincidono, la signature validation fallisce.

Figura 10

3 XML ENCRYPTION

3.1 Panoramica

Insieme a XML Signature, *XML Encryption* rappresenta la principale specifica nell'ambito della sicurezza XML. Lo standard consiste in una metodologia flessibile per la cifratura di documenti arbitrari e per la rappresentazione dei dati cifrati in formato XML. Si basa su molti concetti originariamente definiti in XML Signature. Le sue caratteristiche fondamentali sono:

- la cifratura parziale, che permette di cifrare elementi specifici contenuti in documenti XML;
- la cifratura multipla, che permette di cifrare i dati più di una volta;
- la cifratura complessa, che permette la designazione dei destinatari autorizzati alla decifratura delle rispettive porzioni di dati.

3.2 XML Encryption e specifiche correlate

XML Encryption è stata rilasciata come raccomandazione ufficiale del W3C nel 2002. Il W3C ha anche rilasciato delle specifiche che risolvono i problemi che si presentano quando XML Encryption viene usato in combinazione con XML Signature.

Le specifiche correlate a XML Encryption sono le seguenti:

- XML Encryption Syntax and Processing (Raccomandazione W3C, 2002);
- Decryption Transform for XML Signature (Raccomandazione W3C, 2002).

3.3 XML Encryption Syntax and Processing

La specifica *XML Encryption Syntax and Processing* definisce un formato XML per la rappresentazione di dati cifrati e una serie di regole di processing riguardo le operazione di cifratura e decifratura coinvolte.

XML Encryption si basa su diversi elementi e idee presenti nella sintassi di XML Signature e permette interessanti combinazioni in documenti XML che possono esibire proprietà di firma digitale e cifratura.

XML Encryption condivide elementi, algoritmi e concetti già introdotti nelle specifiche XML Signature. Per questo motivo, nel resto di questo capitolo, si assume che tutti gli elementi mostrati negli esempi appartengano al namespace XML Encryption, a meno che non siano preceduti dal prefisso `ds:` (digital signature), che indica la loro appartenenza al namespace XML Signature.

3.3.1 Principi e sintassi di XML Encryption

A differenza di XML Signature, dove è possibile identificare una singola entità come "una firma XML" con determinate proprietà, XML Encryption si può definire come un processo per cifrare dei dati il cui risultato viene rappresentato usando la sintassi XML. Quindi, XML Encryption può essere visto come una tecnologia di packaging.

Al fine di descrivere le caratteristiche di XML Encryption, si illustreranno preliminarmente due casi d'uso: la cifratura di un flusso di byte arbitrario e quella di dati XML. Ciò perché lo standard prevede delle differenze semantiche tra i due tipi di operazioni. In particolare, in alcuni casi è conveniente eseguire delle operazioni di sostituzione nel documento XML oggetto della cifratura, sostituendo ai dati in chiaro quelli cifrati.

Cifratura di dati arbitrari

Il primo caso d'uso che verrà esaminato riguarda la cifratura di un generico flusso di byte: si consideri una risorsa web espressa per mezzo di un URI alla quale viene applicato un algoritmo crittografico per ottenere una rappresentazione XML dei dati cifrati. La vista di questo processo è mostrata nella figura seguente.

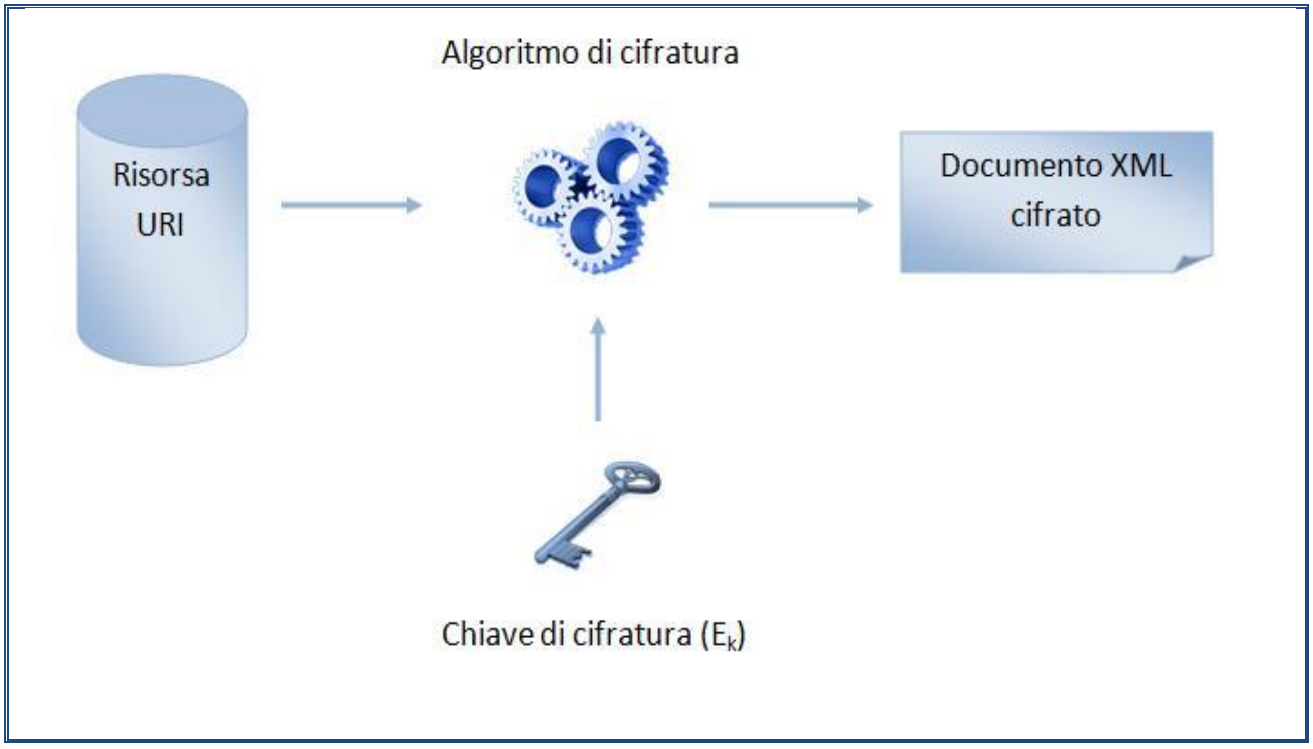


Figura 11

Si tratta di una semplificazione dell'intero processo di cifratura, in cui però sono presenti gli elementi fondamentali. La *Risorsa URI* può essere una risorsa web remota recuperabile attraverso qualche protocollo di comunicazione, oppure un file

locale. L'output dell'algoritmo di cifratura è rappresentato in formato XML. Le caratteristiche che differenziano questo schema di cifratura da uno tradizionale sono il fatto che i dati di input sono specificati come risorsa web e che la rappresentazione dell'operazione di cifratura è in XML (la cui sintassi verrà specificata nel seguito).

I requisiti del documento di output includono alcune delle stesse proprietà richieste da XML Signature: esso deve essere human-readable e fornire la possibilità di recuperare la chiave di decifratura. Inoltre, deve fornire informazioni aggiuntive quali l'algoritmo usato. Tutte queste informazioni sono fornite per mezzo dell'elemento `<EncryptedData>`. La sua struttura è mostrata nella figura seguente.

```
<EncryptedData Id? Type?>
  <EncryptionMethod/>?
  <ds:KeyInfo>
    <EncryptedKey>
    <AgreementMethod>?
    <ds:KeyName>?
    <ds:RetrievalMethod>?
    <ds:*>?
  </ds:KeyInfo>?
  <CipherData>
    <CipherValue>?
    <CipherReference URI?>?
  </CipherData>
  <EncryptionProperties/>?
</EncryptedData>
```

Figura 12

Consideriamo, per ora, una semplice versione di `<EncryptedData>` costituito dal solo elemento `<CipherData>` che contiene il risultato della cifratura. Nella figura seguente viene mostrato un esempio di istanza di tale elemento.

```
<EncryptedData>
  <CipherData>
    <CipherValue>
```

```
      EwJVUzEPMA0GA1UECBMGQXRoZW5zMRUwEwYDVQ
      QKEwxQaGlsb3NvcGhlcnMxETAPBgNVBAMTCFNv
      Y3JhdGVzMSIwIAYJKoZIhvcNAQkBFhNzb2NyYX
      Rlc0BhdGhlbnMuY29tMB4XDTAxMDIxNjIzMjgz
    </CipherValue>
  </CipherData>
</EncryptedData>
```

Figura 13

In questo esempio i dati cifrati sono contenuti nell'elemento `<CipherValue>` figlio di `<CipherData>`. Tali dati sono codificati in Base-64 per fornire una rappresentazione conforme alla sintassi XML dei dati cifrati.

I concetti che emergono nello scenario considerato sono semplici ed intuitivi: esiste un singolo elemento `<EncryptedData>` che contiene i dati cifrati di una sorgente trattata come flusso di byte (il tipo di dati in cui effettivamente consiste è irrilevante).

Cifratura di dati XML

Nel secondo caso d'uso consideriamo un documento XML con dei dati da proteggere. Questi possono essere uno o più elementi con i rispettivi figli (un sottoinsieme del documento), oppure il contenuto di un elemento (solo testo). XML Encryption permette, infatti, la cifratura sia a livello di elemento che a livello di contenuto di elemento.

Quando la sorgente di dati da cifrare consiste in un documento XML, la semantica definita da XML Encryption prevede la cosiddetta "*Plaintext Replacement*", cioè i dati da cifrare vengano sostituiti nel documento originale con la loro versione cifrata. In pratica l'elemento `<EncryptedData>` sostituisce i dati XML in chiaro (un elemento o il suo contenuto), dando origine, a tutti gli effetti, ad un nuovo documento XML.

La figura seguente mostra un esempio di tale meccanismo.

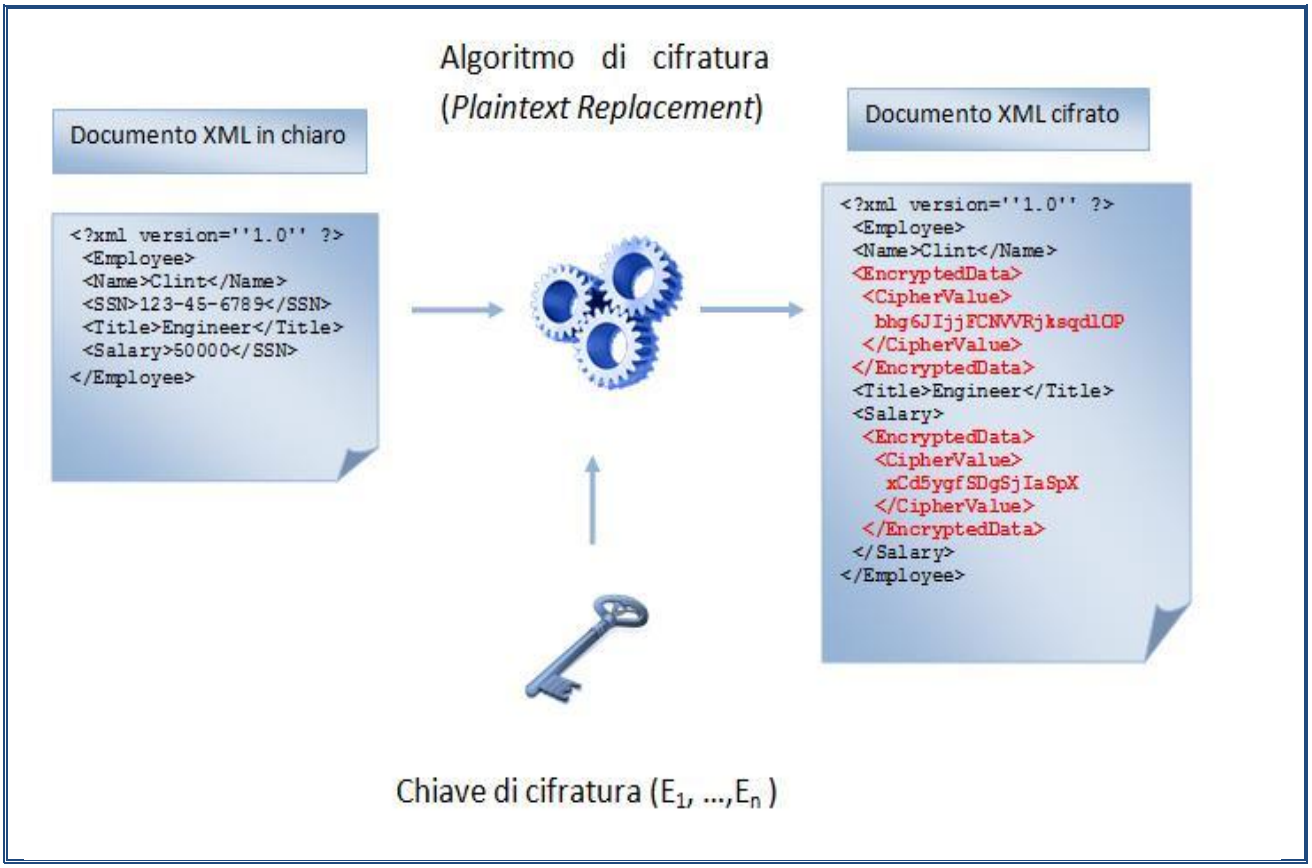


Figura 14

In questo caso il documento XML in input rappresenta il record di un impiegato costituito da quattro elementi: <Name>, <SSN>, <Title> e <Salary>.

```
<?xml version=''1.0'' ?>
<Employee>
  <Name>Clint</Name>
  <SSN>123-45-6789</SSN>
  <Title>Engineer</Title>
  <Salary>50000</SSN>
</Employee>
```

Si assume che i dati da proteggere siano l'intero elemento <SSN> e il contenuto dell'elemento <Salary>. Con XML Encryption, questi dati vengono cifrati e sostituiti con un elemento <EncryptedData>. I dati cifrati appaiono come contenuto testuale codificato in Base-64 dell'elemento <CipherValue>.

```
<?xml version=''1.0'' ?>
<Employee>
  <Name>Clint</Name>
  <EncryptedData>
    <CipherValue>
      bhg6JIjjFCNVVRjksqdlOP
```

```
    </CipherValue>
  </EncryptedData>
  <Title>Engineer</Title>
  <Salary>
    <EncryptedData>
      <CipherValue>
        xCd5ygfSDgSjIaSpX
      </CipherValue>
    </EncryptedData>
  </Salary>
</Employee>
```

Questo semplice esempio tralascia molti dettagli di <EncryptedData>, ma illustra una caratteristica importante: in un singolo documento XML possono apparire diversi elementi <EncryptedData>, ognuno dei quali rappresenta la versione cifrata di un elemento o di un insieme di essi. È, inoltre, possibile cifrare ogni elemento <EncryptedData> con una chiave di cifratura diversa.

Dopo aver discusso questi due casi d'uso, esplicativi del funzionamento basilare di XML Encryption, procederemo descrivendo la sintassi della specifica illustrando gli elementi fondamentali che compongono il blocco <EncryptedData>. Infine saranno presentate le regole di processing stabilite dallo standard per la cifratura (e relativa decifratura) di contenuti usando le tecniche XML Encryption.

Elemento <EncryptedData>

L'elemento <EncryptedData> rappresenta sostanzialmente il contenitore dei dati cifrati e delle informazioni necessarie alla loro decifratura. Può avere quattro possibili figli: <EncryptionMethod>, <KeyInfo>, <CipherData> e <EncryptionProperties>. Inoltre ha i due attributi Id e Type. Il primo rappresenta un identificatore arbitrario, mentre il secondo è usato per l'identificazione del tipo di dati del testo in chiaro. Il termine tipo in questo contesto è da intendersi in senso generico, infatti il valore dell'attributo può sia indicare un tipo di contenuto MIME (nel caso di cifratura di un flusso di byte generico), che uno dei due valori predefiniti Element o Content (nel caso di cifratura di un sottoinsieme di un documento XML).

L'attributo `Type`, benchè opzionale, è di vitale importanza per la corretta elaborazione di un documento XML contenente elementi `<EncryptedData>`. Ad esempio, nel caso di *plaintext replacement*, l'applicazione che effettua la decifratura deve conoscere il tipo di dati del testo in chiaro (Element o Content) allo scopo di ricostruire correttamente il documento XML originale.

Elemento <EncryptionMethod>

L'elemento `<EncryptionMethod>` identifica l'algoritmo di cifratura usato e altre informazioni ausiliare, come uno schema di *key size padding* (per cifrari asimmetrici) o una modalità di cifratura. L'algoritmo di cifratura è denotato con un identificatore URI.

La specifica XML Encryption distingue tra parametri impliciti ed espliciti. I primi includono i dati in chiaro, la chiave di cifratura e il vettore di inizializzazione (IV). I parametri impliciti non sono presenti in nessuno degli elementi o degli attributi di `<EncryptionMethod>` e si assume che l'applicazione sappia come recuperarli (ad esempio il vettore di inizializzazione può essere incluso nel testo cifrato). I parametri espliciti, invece, appaiono come parte dell'identificatore URI di `<EncryptionMethod>` oppure specificati in qualche suo elemento figlio. Essi denotano informazioni quali la dimensione della chiave, la modalità di cifratura, lo schema di padding, o qualsiasi parametro opzionale dipendente dall'applicazione. L'insieme degli algoritmi di cifratura specificati da XML Encryption non è rigida, infatti è possibile aggiungere nuovi algoritmi quando questi divengono disponibili.

L'esempio seguente mostra come si possa specificare l'algoritmo Triple-DES in modalità CBC (Cipher Block Chaining) attraverso `<EncryptionMethod>`:

```
<EncryptionMethod
Algorithm="http://www.w3.org/2001/04/xmlenc#tripledes-cbc"/>
```

In modo simile, l'algoritmo AES (Advanced Encryption Standard) è denotato:

```
<EncryptionMethod
Algorithm=http://www.w3.org/2001/04/xmlenc#aes128-cbc"/>
```

Si può notare come l'identificatore URI contenga parametri espliciti al suo interno. Nel caso del Triple-DES, il suffisso `cbc` denota la modalità omonima. Nel caso AES, il

suffisso `128-cbc` indica, oltre che la modalità CBC, la dimensione del blocco di 128 bit.

L'elemento `<EncryptionMethod>` può avere tre figli: `<KeySize>`, `<ds:DigestMethod>` e `<OAEPparams>`. L'elemento `<Keysize>` è usato per denotare la dimensione della chiave di cifratura, l'elemento `<ds:DigestMethod>` specifica un algoritmo di digest usato per eventuali operazioni di padding, mentre l'elemento `<OAEPparams>` è usato per la trasmissione di parametri necessari per operazioni RSA che usano lo schema OAEP (Optimal Asymmetric Encryption Padding).

Elemento <CipherData>

L'elemento `<CipherData>` (obbligatorio all'interno di `<EncryptedData>`) include o referenzia i dati cifrati. Se funge da contenitore, allora avrà un figlio `<CipherValue>` che contiene i dati cifrati codificati in Base-64. Se invece, `<CipherData>` funge da riferimento, allora conterrà un elemento `<CipherReference>` (la cui logica è simile a `<Reference>` di XML Signature) con un puntatore ai dati cifrati. L'elemento `<CipherReference>` è utile nel caso in cui i dati cifrati siano di grandi dimensioni pregiudicando la loro inclusione in formato testuale direttamente nel documento XML. In questo caso la locazione dei dati cifrati sono specificati da un URI.

Elemento <EncryptionProperties>

L'elemento `<EncryptionProperties>` include alcune proprietà supplementari riguardo i dati cifrati. Esso contiene una lista di elementi `<EncryptionProperty>` che a loro volta possono includere elementi arbitrari provenienti da qualsiasi namespace.

Elemento <KeyInfo>

L'elemento <KeyInfo> corrisponde a quello definito nella specifica XML Signature con l'aggiunta di alcune caratteristiche particolari. Esso descrive come ottenere la chiave per la decifratura del contenuto dell'elemento <CipherData>. Il suo scopo basilare è quindi analogo a quello della sintassi XML Signature. Tuttavia, esiste una importante distinzione. Infatti, per un oggetto firmato, è possibile (e raccomandato) includere la chiave di verifica nell'elemento <Signature>. Ciò perché si assume che il ricevente possa assicurarsi circa la fidatezza della chiave prima di procedere nella verifica della firma. Al contrario, in un oggetto cifrato, la chiave di decifratura non deve essere disponibile (o almeno non lo deve essere in chiaro).

Nella figura seguente viene mostrata la struttura dell'elemento <KeyInfo>.

```
<ds:KeyInfo>
  <EncryptedKey>?
  <AgreementMethod>?
  <ds:KeyName>?
  <ds:RetrievalMethod>?
  <ds:*>?
</ds:KeyInfo>?
```

Figura 15

Un scenario comune per l'elemento <KeyInfo> è utilizzarlo per includere al suo interno una versione cifrata della chiave da usare per la decifratura dei dati. Tale chiave viene salvata nell'apposito elemento <EncryptedKey>. Quest'ultimo ha una struttura simile a quella di <EncryptedData> in quanto deriva dallo stesso tipo <EncryptedType> che fornisce un comportamento basilare per gli elementi che contengono dati cifrati. Nella figura seguente viene mostrata tale relazione.

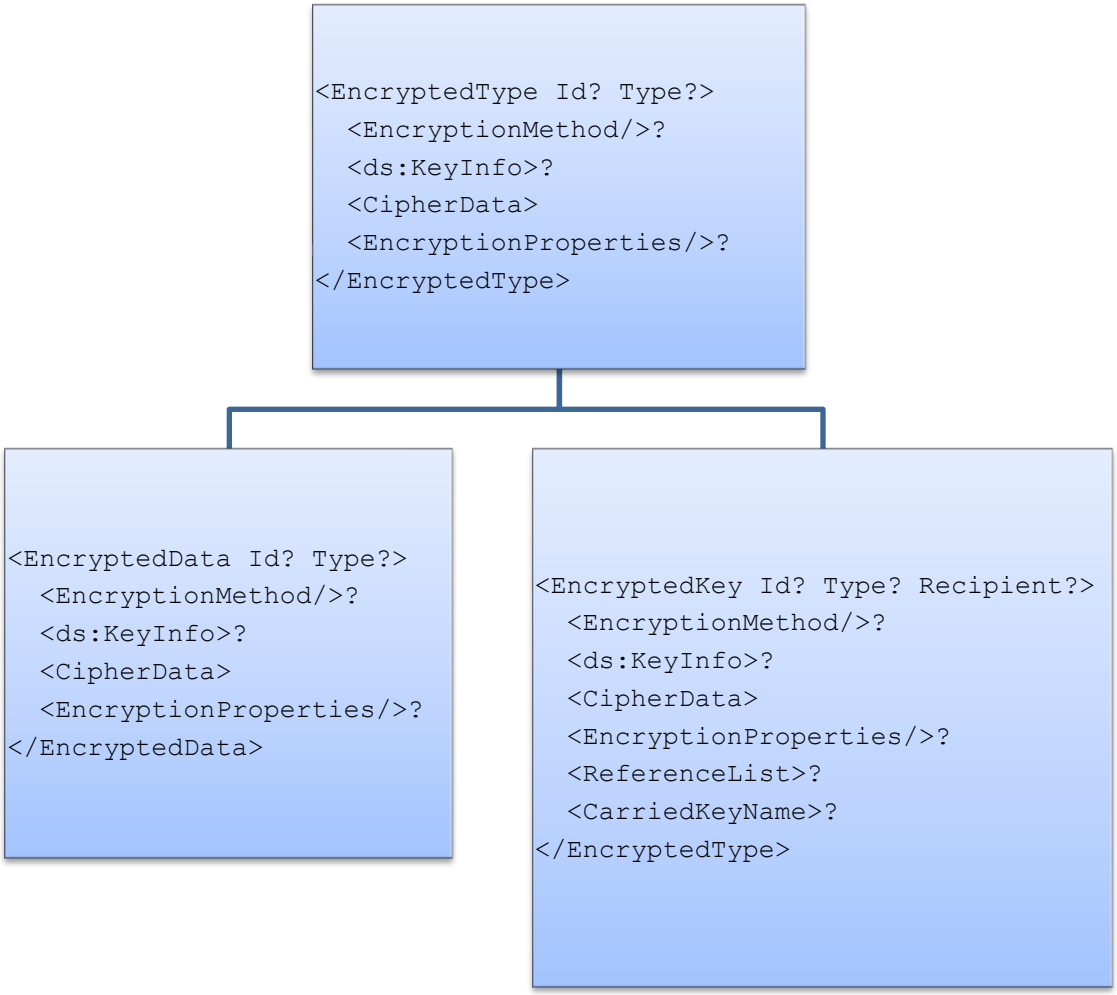


Figura 16

Una proprietà di <EncryptedType> è la possibilità di annidamento ricorsivo. Il classico esempio è quello che si verifica quando un elemento <EncryptedData> ne contiene uno <EncryptedKey> (all'interno di <KeyInfo>). Tuttavia è possibile anche che un elemento <EncryptedKey> sia annidato in un altro elemento <EncryptedKey> (il caso più comune è quello della "*digital envelope*", dove viene usato un cifrario simmetrico per cifrare i dati e l'algoritmo RSA per la cifratura della chiave).

La figura seguente mostra un esempio di dati cifrati con XML Encryption utilizzando l'algoritmo AES con una chiave di 128 bit.

```
<EncryptedData xmlns="http://www.w3.org/2001/04/xmlenc#"
     Type="http://www.isi.edu/in-notes/iana/assignments/
```

```
    media-types/text/html">
  <EncryptionMethod
    Algorithm="http://www.w3.org/2001/04/xmlenc#aes128-cbc"/>
  <ds:KeyInfo xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
    <EncryptedKey>
      <EncryptionMethod
       Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-1_5"/>
      <ds:KeyInfo xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
        <ds:KeyName> John Q. Recipient </ds:KeyName>
      </ds:KeyInfo>
      <CipherData>
        <CipherValue>
          zc1msFhM1GKMYDgYQAAoGActA8YG
        </CipherValue>
      </CipherData>
    </EncryptedKey>
  </ds:KeyInfo>
  <CipherData>
    <CipherValue>
      NFNNJoqMcT2ZfCPrfvYvQ2jRzBFMB4GA1UdEQQXMBWBE2
    </CipherValue>
  </CipherData>
</EncryptedData>
```

Figura 17

In questo caso il tipo di dati è "`text/html`", si tratta quindi della cifratura di un documento HTML. La chiave usata per cifrare i dati viene a sua volta cifrata (è rappresentata dalla struttura `<EncryptedKey>` mostrata in rosso). Tale chiave viene cifrata usando l'algoritmo RSA con padding PKCS #1. Inoltre, la chiave RSA usata per cifrare la chiave AES è la chiave pubblica di *John Q. Recipient* (specificato in `<KeyName>`) il quale dovrebbe essere l'unico destinatario in grado di decifrare il messaggio. Ovviamente, in uno scenario reale l'identificatore usato in `<KeyName>` dovrebbe essere meno generico e permettere l'identificazione univoca del destinatario.

Un importante elemento di `<KeyInfo>` è `<AgreementMethod>` che viene utilizzato per supportare l'accordo su un valore segreto condiviso. La specifica XML Encryption supporta solamente l'accordo sulla chiave di tipo asincrono. Ciò significa che mittente e destinatario che partecipano all'accordo genereranno le loro chiavi in tempi differenti. Questo contrasta con un accordo sulle chiavi che potrebbe essere usato in un protocollo di comunicazione, dove generalmente una chiave viene negoziata simultaneamente dalle parti.

3.3.2 XML Encryption Processing

La specifica XML Encryption stabilisce, oltre alla sintassi XML, delle regole di processing che le implementazioni dello standard devono seguire nel processo di cifratura e decifratura. Allo scopo, la specifica introduce tre entità:

- ***Application*** – corrisponde all'entità applicativa che usufruisce delle funzionalità di una data implementazione di XML Encryption, fornendo ad essa i dati e i parametri necessari alle operazioni crittografiche.
- ***Encryptor*** – è l'entità applicativa che esegue le operazioni di cifratura. Esso è responsabile della generazione e della corretta combinazione di tutti gli elementi della sintassi XML Encryption inclusi chiavi, dati cifrati e attributi.
- ***Decryptor*** – è l'entità applicativa che esegue la funzione opposta all'*encryptor*. È responsabile, cioè, della decodifica e della decifratura dei dati criptati.

Sulla base di queste entità, nel seguito verranno mostrati i passi, definiti nello standard, coinvolti nelle operazioni di cifratura e decifratura.

Cifratura

Per ogni oggetto di dati da cifrare, l'*encryptor* deve:

1. Selezionare l'algoritmo (e i relativi parametri) da utilizzare per la cifratura.
2. Ottenere e (opzionalmente) rappresentare la chiave:
   - Se la chiave deve essere identificata si costruisce l'appropriato elemento `<KeyInfo>`.
   - Se la chiave deve essere a sua volta cifrata, si costruisce un elemento `<EncryptedKey>` applicando ricorsivamente il processo di cifratura corrente.
3. Cifrare i dati:
   - Se i dati corrispondono ad un elemento XML (oppure al suo contenuto), si ottiene il flusso di byte corrispondente serializzandoli in UTF-8 (la serializzazione può anche essere eseguita dall'*application*).

- Se i dati sono di un altro tipo e non sono sottoforma di flusso di byte, l'*application* deve serializzarli.
- Si cifra il flusso di byte ottenuto con l'algoritmo e la chiave dei passi 1 e 2.
- Se il *decryptor* non è in grado di dedurre implicitamente il tipo di dati, l'*encryptor* deve fornirlo (ad esempio, attraverso gli attributi `MimeType` o `Encoding`).

4. Costruire la struttura `<EncryptedType>` (`<EncryptedData>` o `<EncryptedKey>`) con tutte le informazioni prodotte nei punti precedenti:
   - Se il flusso di byte cifrato ottenuto al passo 3 viene incluso nell'elemento `<CipherData>`, allora deve essere codificato in Base-64 e inserito come contenuto di `<CipherValue>`.
   - Se, invece, il flusso di dati cifrato è salvato esternamente alla struttura `<EncryptedType>`, devono essere fornite l'URI e le trasformazioni (se necessarie) nell'elemento `<CipherReference>`.
5. Elaborare `<EncryptedData>`:
   - Se i dati cifrati sono un elemento XML (o il suo contenuto), l'*encryptor* deve restituire l'elemento `<EncryptedData>` all'*application*. Quest'ultima può usarlo come elemento top level in un nuovo documento XML o inserirlo in un altro documento XML.

     In caso di *plaintext replacement*, l'*encryptor* dovrebbe essere in grado di sostituire l'elemento da cifrare con l'elemento `<EncryptedData>` che contiene la versione cifrata: quando un'*application* richiede la sostituzione di un elemento, fornisce il documento XML e l'identificazione dell'elemento da sostituire all'*encryptor* il quale rimuove gli elementi richiesti e li sostituisce con i corrispondenti `<EncryptedData>`.
   - Se i dati cifrati non sono un elemento XML (o il suo contenuto), l'*encryptor* deve restituire l'elemento `<EncryptedData>` all'*application* che può usarlo come elemento top level in un nuovo documento XML o inserirlo in un altro documento XML.

Decifratura

Per ogni elemento derivato da `<EncryptedType>` (`<EncrypteData>` o `<EncryptedKey>`) che deve essere decifrato, il *decryptor* deve:

1. Processare l'elemento per determinare l'algoritmo (con i relativi parametri) da usare. Se qualche informazione è omessa, deve essere fornita dall'*application*.
2. Localizzare la chiave per la decifratura in accordo alle informazioni contenute nell'elemento `<KeyInfo>`.
3. Decifrare i dati contenuti nell'elemento `<CipherData>`:
   - Se è presente l'elemento `<CipherValue>`, allora ne viene recuperato il contenuto decodificandolo in Base-64.
   - Se è presente un elemento `<CipherReference>`, viene utilizzato l'URI e le trasformazioni specificate (se previste) per recuperare il flusso di byte.
   - Il flusso di byte criptato viene decifrato utilizzando l'algoritmo e la chiave determinati ai passi 1 e 2.
4. Processare i dati decifrati di tipo elemento XML (o suo contenuto):
   - Il flusso di byte corrispondente ai dati in chiaro ottenuto al passo 3 viene interpretato come insieme di caratteri codificati in UTF-8.
   - Il *decryptor* deve restituire all'*application* i dati decifrati.
   - Il *decryptor* dovrebbe essere capace di sostituire nel documento XML l'elemento `<EncryptedData>` con l'elemento in chiaro ottenuto con la decifratura.
5. Processare i dati che non sono di tipo elemento XML (o suo contenuto).
   - Il flusso di byte decifrato deve essere restituito all'*application* corredato delle informazioni specificate negli attributi `Type`, `MimeType` e `Encoding`.

3.4 Decryption Transform for XML Signature

Lo scopo della specifica *Decryption Transform for XML Signature* è quello di fornire un supporto alla gestione dei problemi che possono verificarsi quando le tecniche di cifratura e firma digitale fornite da XML Encryption e XML Signature sono usate contemporaneamente sullo stesso documento (un caso molto comune).

Una caratteristica di XML Signature è quella di assicurare l'integrità di un documento in modo che qualsiasi alterazione possa essere rilevata. Tuttavia, molte applicazioni richiedono la possibilità di poter prima firmare un documento XML e quindi cifrare

parte di esso, alterandone così il contenuto. La Decryption Transform permette al ricevente di stabilire quali parti del documento decifrare in modo da riportarlo allo stato originale prima di verificare la firma.

Per illustrare in dettaglio le motivazioni che hanno portato allo sviluppo di questa specifica, consideriamo un documento XML che contenga sia un elemento `<Signature>` che uno (o più) `<EncryptedData>`. In tale situazione, è possibile che, decifrando l'elemento `<EncryptedData>`, venga invalidata la firma XML Signature. Ciò succede se l'elemento `<EncryptedData>` viene firmato con XML Signature. Infatti una volta che `<EncryptedData>` viene decifrato e, attraverso la *plaintext replacement*, sostituito con l'elemento in chiaro, il documento originale risulta alterato facendo così fallire la verifica della firma.

Esaminiamo questo scenario con un esempio: la figura seguente mostra un documento XML che deve essere firmato e cifrato.

```
<SecureDoc>
  <SensitiveInfo1>
    Send 5 million to the North base!
  </SensitiveInfo1>
  <SensitiveInfo2>
    Attack at dawn!
  </SensitiveInfo2>
</SecureDoc>
```

Figura 18

Supponiamo di cifrare il contenuto dell'elemento `<SensitiveInfo1>` e di sostituirlo con il corrispondente `<EncryptedData>`. Il risultato di questa operazione è mostrato nella figura seguente.

```
<SecureDoc>
  <SensitiveInfo1>
    <EncryptedData Id="ED1"
                   Type="http://www.w3.org/2001/04/xmlenc#Content"
                   xmlns="http://www.w3.org/2001/04/xmlenc#">
```

```
      <EncryptionMethod Algorithm="http:
                        //www.w3.org/2001/04/xmlenc#tripledes-cbc"/>
      <CipherData>
        <CipherValue>
          qF1z+e5Jwvy49vVmZpMkb/3aMdr4ESGmTbc7FcQ
        </CipherValue>
      </CipherData>
    </EncryptedData>
  </SensitiveInfo1>
  <SensitiveInfo2>
    Attack at dawn!
  </SensitiveInfo2>
</SecureDoc>
```

Figura 19

Creiamo, ora, una firma XML di tipo enveloping sull'intero documento così ottenuto. La figura seguente mostra il risultato dell'operazione di firma.

```
<Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
  <SignedInfo>
    <CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/
                                       REC-xml-c14n-20010315"/>
    <SignatureMethod Algorithm="http://www.w3.org/2000/
                                09/xmldsig#rsa-sha1"/>
    <Reference URI="#envelopedData">
      <DigestMethod Algorithm="http://www.w3.org/
                               2000/09/xmldsig#sha1"/>
      <DigestValue>3NMzvYIWFHiF3LStTgxtQkS9NpI=</DigestValue>
    </Reference>
  </SignedInfo>
  <SignatureValue>
    F2dXbU267Zaw/bsDfpM4GkqbDIl7JcdU6mR+
    yYtvEFAK87v6j5vyf8X8TF0HWqMK
    BPlvQthSFBEcKurEkHxcvQ==
  </SignatureValue>
  <ds:Object Id="envelopedData" xmlns=""
             xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
    <SecureDoc>
      <SensitiveInfo1>
        <EncryptedData Id="ED1"
                   Type="http://www.w3.org/2001/04/xmlenc#Content"
                   xmlns="http://www.w3.org/2001/04/xmlenc#">
          <EncryptionMethod Algorithm="http://www.w3.org/2001/04
                                       /xmlenc#tripledescbc"/>
          <CipherData>
```

```
            <CipherValue>
              qF1z+e5Jwvy49vVmZpMkb/3aMdr4ESGmTbc7FcQ
            </CipherValue>
          </CipherData>
        </EncryptedData>
      </SensitiveInfo1>
      <SensitiveInfo2>
        Attack at dawn!
      </SensitiveInfo2>
    </SecureDoc>
  </ds:Object>
</Signature>
```

Figura 20

A questo punto, si presenta un problema sull'ordine con cui eseguire le operazioni al lato ricevente. Cioè, se il destinatario decide di decifrare i dati criptati prima di verificare la firma, si avrà una sostituzione dell'elemento `<EncryptedData>` in `<SecureDoc>` e, pertanto, il valore digest dell'elemento `<Reference>` sarà errato.

Consideriamo, inoltre, cosa succede se modifichiamo il contenuto di `<Object>` cifrando (ma senza firmare) l'elemento `<SensitiveInfo2>`. Nella figura seguente viene mostrato questo scenario.

```
<ds:Object Id="envelopedData" xmlns=""
                    xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
  <SecureDoc>
    <SensitiveInfo1>
      <EncryptedData Id="ED1"
                  Type="http://www.w3.org/2001/04/xmlenc#Content"
                  xmlns="http://www.w3.org/2001/04/xmlenc#">
        <EncryptionMethod Algorithm="http://www.w3.org/2001/
                                     04/xmlenc#tripledes-cbc"/>
        <CipherData>
          <CipherValue>
            qF1z+e5Jwvy49vVmZpMkb/3aMdr4ESGmTbc7FcQ
          </CipherValue>
        </CipherData>
      </EncryptedData>
    </SensitiveInfo1>
    <SensitiveInfo2>
      <EncryptedData Id="ED2"
```

```
                Type="http://www.w3.org/2001/04 /xmlenc#Content"
                xmlns="http://www.w3.org/2001/04 /xmlenc#">
        <EncryptionMethod Algorithm="http://www.w3.org/2001/04
                                     /xmlenc#tripledes-cbc"/>
        <CipherData>
          <CipherValue>
            tvEFAK87v6j5vyf8X8TF0HWqMKBPlvQthSFBEcKu
          </CipherValue>
        </CipherData>
      </EncryptedData>
    </SensitiveInfo2>
  </SecureDoc>
</ds:Object>
```

Figura 21

Adesso il documento è più complesso: l'elemento `<Object>` contiene due elementi `<EncryptedData>`, e un possibile destinatario non ha modo di conoscere l'ordine con cui sono state eseguite le operazioni di cifratura e di firma. Cioè, se il ricevente prova a validare la firma XML, avrà un insuccesso a causa del fatto che tale firma è stata calcolata su uno solo dei due elementi `<EncryptedData>`.

La Decryption Transform permette di specificare una lista di elementi `<EncryptedData>` che la cui decifratura deve essere tralasciata per preservare una corretta verifica della firma. Nell'esempio considerato, quindi, per la corretta verifica della firma, deve essere specificato al ricevente che l'elemento `<EncryptedData>` il cui attributo `Id` ha valore `ED1` non deve essere decifrato, mentre tutte le altre istanze di `<EncryptedData>` devono essere decifrate.

4 Bibliografia

- **Security, Privacy, and Trust in Modern Data Management**; Series: Data-Centric Systems and Applications; Petkovic, Milan; Jonker, Willem (Eds.) 2007.

- **XML Security**, Blake Dournaee, McGraw-Hill, 2002.

- **Introduction to XML Encryption and XML Signature**; Berin Lautenbach; Information Security Technical Report, Vol. 9, Issue 3, pagine 6-18, 2004.

- **XML Security**; E.Bertino, B.Carminati, E.Ferrari; Information Security Technical Report, Vol. 6, Issue 2, pagine 44-58, 2001.

- **XML Signature/Encryption-the Basis of Web Services Security**; Miyauchi Koji; Nec J Adv Technol, Vol. 2, N. 1, pagine 35-39, 2005.

Specifiche del W3C:

- ***XML Signature Syntax and Processing***
  http://www.w3.org/TR/xmldsigcore/

- ***XML Encryption Syntax and Processing***
  http://www.w3.org/TR/xmlencryptionreq