Studi dan Implementasi XML *Signature* dalam Microsoft .NET

Diko Aldillah Patiwiri – NIM : 13503046

Program Studi Teknik Informatika, Institut Teknologi Bandung
Jl. Ganesha 10, Bandung
E-mail : if13046@students.if.itb.ac.id

Abstraksi

Security saat ini telah menjadi perhatian utama dalam pembangunan berbagai jenis aplikasi, terutama ketika terjadi pertukaran data atau informasi yang penting, rahasia, dan kritis dengan lingkungan luar sistem. Terdapat beberapa cara untuk mengirimkan dan menerima data secara aman, antara lain dengan menggunakan **HTTPS** dan kriptografi kunci public (*Public Key Cryptography*). Dalam perkembangan teknologi saat ini, XML telah menjadi sebuah media standar dalam komunikasi data antara berbagai aplikasi. Spesifikasi XML *Signature* yang ada saat ini yang dikeluarkan oleh W3C, telah memungkinkan pengiriman dan penerimaan data secara aman, hal ini dikarenakan format XML saat ini mendukung kompatibilitas dengan berbagai jenis metode/algoritma yang sudah ada seperti PGP, RSA, DSA, dan lain-lain.

Makalah ini akan coba membahas mengenai studi dan implementasi XML *Signature* di dalam lingkungan Microsoft .NET. XML *Signature* yang akan dibahas pada makalah ini mengacu pada standar yang dibuat oleh W3C. Sebuah perangkat lunak yang bernama **XJS** (**XML Joga Signature**) akan dibangun untuk mendukung pengimplementasian XML *Signature*. **XJS** ini dikembangkan dengan menggunakan bahasa pemrograman Visual C#. Di dalam lingkungan .NET itu sendiri telah disediakan *classes* dan *methods* untuk melakukan proses penandatanganan dokumen XML secara digital yaitu dengan menggunakan **System.Security.Cryptography.Xml** *namespace*.

Perangkat lunak ini kemudian akan digunakan untuk melakukan pembangkitan sepasang kunci (*keypair*) dengan menggunakan algoritma DSA, yaitu *PublicKey* dan *PrivateKey*. Selain itu dapat pula dilakukan pembangkitan hanya terhadap *PublicKey*. Kunci-kunci ini nantinya akan disimpan dalam sebuah file XML. Dengan menggunakan *PrivateKey* ini nantinya akan dilakukan proses komputasi terhadap sebuah dokumen XML dan di-*generate* sebuah struktur XML yang telah ter-*signed* secara digital. Struktur XML yang dihasilkan ini sesuai dengan spesifikasi yang dikeluarkan oleh W3C. Perangkat lunak ini juga nantinya dapat melakukan verifikasi terhadap *signature* dalam sebuah *signed* XML document dengan menggunakan *PublicKey*. Otentikasi/verifikasi ini menghasilkan keterangan apakah terjadi modifikasi terhadap dokumen tersebut.

***Kata kunci**:* XML *Signature*, Microsoft .NET, Visual C#, W3C, *Public Key, Private Key*

1. Pendahuluan

Ketika terdapat sebuah kebutuhan dalam pengaksesan data dari sebuah aplikasi dengan aplikasi eksternal yang lain, maka dalam hal ini *security* memegang peranan yang sangat penting. Aplikasi tersebut harus menjaga informasi-informasi agar tidak terusik selama transmisinya. Contoh solusi yang telah banyak dipakai adalah dengan memanfaatkan HTTPS dan kriptografi kunci publik. HTTPS memberikan *contraints* dalam hal performansi, sedangkan kriptografi kunci publik menyaratkan kepada pengirim dan penerima agar menggunakan sebuah algoritma yang sama dan spesifik.

W3C telah merilis sebuah standar baru yang disebut XML *Signature*, yang berfungsi menandatangani sebuah dokumen XML secara digital. Dibuatnya standar ini telah mendatangkan sebuah kemudahan, karena banyak aplikasi-aplikasi yang dibuat dengan bahasa berbasis Java ataupun Microsoft .NET telah mendukung `SignedXML` dan dapat

memvalidasi *signature*. Microsoft .NET *framework* sendiri telah mempunyai sebuah *namespace* yang secara spesifik telah mendukung XML *Signature*.

Berikut adalah perbedaan antara *Digital Signature* dan enkripsi kunci publik :

Table 1. Perbedaan enkripsi kunci publik dan digital signature

Enkripsi kunci publik	***Digital Signature***
Keseluruhan isi data dienskripsi	Hanya *MessageDigest* yang dienkripsi
Kunci publik digunakan untuk mengenkripsi data sedangkan kunci privat digunakan untuk mendekripsikan data	Kunci privat digunakan untuk mengenkripsi *MessageDigest* dan kunci publik digunakan dalam dekripsi untuk mendapatkan DigestValue dan menghitung kembali MessageDigest untuk diperiksa apakah data telah mengalami modifikasi

Dari tabel diatas menggambarkan informasi dasar tentang enkripsi kunci publik dan *Digital Signature*. Jika terdapat sebuah situasi dimana informasi dari satu perusahaan ke beberapa perusahaan lain melalui **FTP**, maka enkripsi menggunakan kriptografi kunci publik adalah solusi yang sangat ideal. Namun jika isi data tidak terlalu rahasia namun masih diperlukan untuk diotentikasi untuk keperluan verifikasi, maka *digital signature* sebaiknya yang dipilih.

XML *Signature* mendukung penggunaan beberapa algoritma seperti RSA, DSA, PGP, dan lain-lain, juga standar-standar yang dikeluarkan oleh W3C. Dalam bab-bab berikutnya akan coba dibahas tentang seluk beluk XML Signature, bagaimana XML *Signature* dapat digunakan, bagaimana Microsoft .NET menyediakan fungsionalitas untuk meng-*sign* sebuah dokumen XML, dan implementasi XML *Signature* dalam Microsoft .NET.

2. XML Signature

XML *Signature* dapat diimplementasikan pada beberapa jenis konten digital (*data object*), termasuk XML. Sebuah XML *Signature* dapat diaplikasikan terhadap konten dalam suatu *resource* atau lebih. Pada dasarnya XML *Signature* adalah suatu dokumen XML yang terdiri dari elemen-elemen dasar pada dokumen XML sebelumnya dan tanda tangan (*signature*) itu sendiri sebagai salah satu elemen di dalamnya. Terdapat sebuah *signature* yang dikenal dengan *enveloped signature* dalam dokumen XML yang sama yang berperan dalam perhitungan *SignatureValue*. Namun *enveloped signature* ini harus dapat memastikan bahwa nilai yang terkandung di dalamnya ini tidak masuk ke dalam perhitungan.

Terdapat pula *signature* yang disebut dengan *detached signature*. Jenis *signature* ini merupakan bagian yang terpisah dengan elemen *signature* dalam dokumen XML, dan dapat diidentifikasi melalui sebuah **URI** atau **TRANSFORM**. *Signature* disertakan dengan dokumen yang ditandatangani. *Signature* ini dipakain untuk memisahkan objek-objek data, juga termasuk sebuah *instance* dimana **Signature** dan objek data berada dalam dokumen XML yang sama namun keduanya merupakan elemen yang saling berkaitan.

2.1. Struktur XML *Signature*

XML *Signature* diaplikasikan pada konten digital (*data object*) secara acak, yaitu dilakukan *hashing* pada data *object* (*DigestValue*). Berdasarkan spesifikasi yang dikeluarkan W3C, struktur XML *signature* dapat dilihat pada kode XML dibawah (dimana "?" menandakan nol atau satu kemunculan, "+" menandakan satu atau lebih kemunculan, dan "*" menandakan nol atau lebih kemunculan).

Kode XML 1. Struktur XML *Signature*

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

Spesifikasi standar diatas digunakan pada sebuah *SignedXML* yaitu dalam pencarian nilai dari *SigningKey*, *SignatureValue*, *DigestValue*, dan *Data*. Pada bab yang selanjutnya akan dibahas spesifikasi XML ini lebih lanjut dan implementasi penanda tanganan dokumen XML dalam Microsoft .NET, yaitu dengan menggunakan bahasa Visual C#.

Signature berhubungan dengan *data object* melalui **URI** (*Uniform Resource Identifiers*). Di dalam sebuah dokumen XML, *signatures* berhubungan dengan beberapa *data object* lokal melalui *fragment identifiers*. Lokal data dapat disertakan dalam sebuah *enveloping signature* atau dapat dimasukkan dalam sebuah *enveloped signature*. *Detached signature* berada diluar *data objects* lokal yang berada dalam dokumen XML yang sama sebagai elemen yang berkaitan. Dalam kasus ini, *signature* bukan jenis *enveloping* (*signature* sebagai *parent*) ataupun *enveloped* (*signature* sebagai *child*).

2.2. Contoh *Signature*

Contoh dibawah ini adalah sebuah *detached signature* pada konten HTML4 dalam standar spesifikasi XML.

Kode XML 2. *Detached Signature*

```
 [s01] <Signature Id="MyFirstSignature"
xmlns="http://www.w3.org/2000/09/xmldsig#">
 [s02]   <SignedInfo>
 [s03]   <CanonicalizationMethod
Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
 [s04]   <SignatureMethod
Algorithm="http://www.w3.org/2000/09/xmldsig#dsa-sha1"/>
 [s05]   <Reference
URI="http://www.w3.org/TR/2000/REC-xhtml1-20000126/">
 [s06]     <Transforms>
 [s07]       <Transform Algorithm="http://www.w3.org/TR/2001/REC-xml-
14n-20010315"/>
 [s08]     </Transforms>
 [s09]     <DigestMethod
lgorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
 [s10]     <DigestValue>j6lwx3rvEPO0vKtMup4NbeVu8nk=</DigestValue>
 [s11]   </Reference>
 [s12] </SignedInfo>
 [s13]   <SignatureValue>MC0CFFrVLtRlk=...</SignatureValue>
 [s14]   <KeyInfo>
 [s15a]    <KeyValue>
 [s15b]      <DSAKeyValue>
 [s15c]        <P>...</P><Q>...</Q><G>...</G><Y>...</Y>
 [s15d]      </DSAKeyValue>
 [s15e]    </KeyValue>
 [s16]   </KeyInfo>
 [s17] </Signature>
```

a. `[s02-12]`:
Elemen <`SignedInfo`> berisi informasi yang ditandatangani. Terdapat dua proses utama dalam validasi <`SignedInfo`>, validasi *signature* pada <`SignedInfo`> dan validasi tiap *digest reference* dalam <`SignedInfo`>. Harus diketahui bahwa algoritma yang dipakai dalam perhitungan <`SignatureValue`> juga disertakan dalam informasi yang di-*signed*, dan juga elemen <`SignatureValue`> berada di luar <`SignedInfo`>.

b. `[s03]` : <`CanonicalizationMethod`> adalah algoritma yang dipakai untuk meng-*canonicalize* elemen <`SignedInfo`> sebelum dilakukan di-*digest* sebagai bagian dari operasi *signature*. Harus diketahui bahwa contoh diatas dan contoh-contoh yang lain dalam makalah ini bukan dalam bentuk *canonical*.

c. `[s04]` :

<SignatureMethod> adalah algoritma yang digunakan untuk mengkonversikan <SignedInfo> yang telah ter-*canonicalize* menjadi <SignatureValue>. Algoritma ini merupakan kombinasi dari algoritma *digest*, *key dependent*, dan mungkin juga algoritma lain seperti *padding*, contohnya **RSA-SHA1**.

d. [s05-11] :
Setiap elemen <Reference> menyertakan *digest method* dan menghasilkan nilai *digest* terhadap *data object* yang teridentifikasi. Juga mungkin disertakan *transform* yang menghasilkan masukan bagi operasi *digest*. Sebuah *data object* di-*sign* dengan menghitung *digest value*-nya dan sebuah *signature* pada nilai tersebut. *Signature* nantinya diperiksa dengan menggunakan *reference* dan *signature validation*.

e. [s14-16] :
<KeyInfo> menggambarkan kunci yang digunakan untuk memvalidasi *signature*. Hal-hal yang mungkin dibutuhkan dalam proses identifikasi antara lain sertifikat, nama kunci-kunci, dan algortma dan informasi kunci yang disepakati. Pemakaian <KeyInfo> bersifat *opsional*, hal ini dikarenakan dua alasan. Pertama, *signer* mungkin tidak memberikan informasi kunci pada semua *processing* dokumen. Kedua, informasi mungkin hanya dibutuhkan dalam konteks aplikasi saja dan tidak perlu direpresentasikan secara eksplisit. Karena <KeyInfo> berada diluar <SignedInfo>, sehingga jika *signer* ingin menyertakan informasi kunci dalam *signature*, maka sebuah <Reference> dapat dengan mudah mengidentifikasi dan menyertakan <KeyInfo> sebagai bagian dari *signature*.

3. Langkah-Langkah Penyusunan XML *Signature*

Secara garis besar, langkah-langkah pembentukan XML *Signature* adalah sebagai berikut :

3.1. Tentukan dokumen yang akan ditandatangani

Informasi mengenai dokumen yang akan ditandatangani akan disediakan dalam bentuk **URI** (*Uniform Resource Identifiers*, contoh :

- http://www.thesite.com/index.html : mengacu pada sebuah halaman HTML di web.
- http://www.thesite.com/logo.gif : mengacu pada sebuah citra GIF di web.
- http://www.thesite.com/tes.xml : mengacu pada sebuah file XML di web.
- http://www.thesite.com/tes.xml#sender : mengacu pada suatu elemen spesifik pada sebuah file XML di web.

3.2. Lakukan perhitungan nilai digest untuk setiap dokumen

Pada XML *Signature*, setiap dokumen yang diacu (*reference*) dispesifikasikan dengan menggunakan elemen <Reference>. Nilai *digest* untuk *reference* tersebut dispesifikasikan dengan penggunaan elemen <DigestValue> sebagai elemen *child* dari elemen <Reference>. Elemen <DigestMethod> digunakan untuk menghitung nilai *digest*. Sebagai contoh perhatikan kode XML *Signature* dibawah ini.

Kode XML 3. Perhitungan nilai digest XML *Signature*

```
<Reference URI="http://www.abccompany.com/news/2000/03_27_00.htm">
<DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
<DigestValue>j6lwx3rvEPO0vKtMup4NbeVu8nk=</DigestValue>
</Reference>
<Reference URI="http://www.w3.org/TR/2000/WD-xmldsig-core-
20000228/signature-example.xml">
<DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
<DigestValue>UrXLDLBIta6skoV5/A8Q38GEw44=</DigestValue>
</Reference>
```

3.3. Satukan elemen reference

Satukan elemen <Reference> (beserta nilai *digest* masing-masing) dalam elemen

<SignedInfo> seperti contoh pada kode XML dibawah ini :

Kode XML 4. Penyatuan elemen `Reference`

```
<SignedInfo Id="foobar">
<CanonicalizationMethod
Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
<SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#dsa-sha1"
/>
<Reference URI="http://www.abccompany.com/news/2000/03_27_00.htm" />
<DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
<DigestValue>j6lwx3rvEPO0vKtMup4NbeVu8nk=</DigestValue>
</Reference>
<Reference URI="http://www.w3.org/TR/2000/WD-xmldsig-core-
20000228/signature-example.xml">
<DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
<DigestValue>UrXLDLBIta6skoV5/A8Q38GEw44=</DigestValue>
</Reference>
</SignedInfo>
```

Kode diatas merupakan perluasan dari kode pada kode pada upa bab 3.2 diatas. Elemen <CanonicalizationMethod> memberikan informasi mengenai algoritma yang digunakan untuk proses kanonikalisasi. Proses kanonikalisasi akan dibahas pada upa bab selanjutnya. Elemen <SignatureMethod> mengidentifikasikan algoritma yang digunakan untuk mendapatkan nilai *signature*.

3.4. Penandatanganan

Hitung nilai *digest* dari elemen <SignedInfo>, tandatangani nilai *digest* tersebut dan simpan nilai *signature* pada elemen <SignatureValue> seperti dapat dilihat pada kode dibawah ini :

Kode XML 5. Penandatanganan

```
<SignatureValue>MC0E
LE=</SignatureValue>
```

3.5. Tambahkan informasi kunci

Letakkan informasi tambahan tentang kunci pada elemen <KeyInfo>. Informasi tambahan ini berupa kunci publik yang dapat digunakan untuk proses verifikasi. Salah satu contoh informasi kunci yang digunakan adalah X509 seperti kode dibawah ini :

Kode XML 6. Tambah informasi kunci

```
<KeyInfo>
<X509Data>
<X509SubjectName>CN=Ed Simon,
O=XMLSecInc.,ST=OTTAWA,C=CA
</X509SubjectName>
<X509Certificate>MIID5jCCA0
+gA...lVN</X509Certificate>
</X509Data>
</KeyInfo>
```

3.6. Tutup XML *Signature* dengan elemen `Signature`

Letakkan elemen <SignedInfo>, <SignatureValue>, dan <KeyInfo> dalam elemen <Signature>. Kode XML *Signature* secara keseluruhan dapat dilihat pada kode dibawah ini :

Kode XML 7. Penutupan XML *Signature*

```
<?xml version="1.0" encoding="UTF-8"?>
<Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
<SignedInfo Id="foobar">
<CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-
xmlc14n-
20010315"/>
<SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#dsasha1"
```

```
/>
<Reference URI="http://www.abccompany.com/news/2000/03_27_00.htm">
<DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
<DigestValue>j6lwx3rvEPO0vKtMup4NbeVu8nk=</DigestValue>
</Reference>
<Reference URI="http://www.w3.org/TR/2000/WD-xmldsig-core-
20000228/signature-example.xml">
<DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
<DigestValue>UrXLDLBIta6skoV5/A8Q38GEw44=</DigestValue>
</Reference>
</SignedInfo>
<SignatureValue>MC0E~LE=</SignatureValue>
<KeyInfo>
<X509Data>
<X509SubjectName>CN=Ed Simon,O=XMLSec
Inc.,ST=OTTAWA,C=CA</X509SubjectName>
<X509Certificate>MIID5jCCA0+gA...lVN</X509Certificate>
</X509Data>
</KeyInfo>
</Signature>
```

4. Proses Verifikasi XML *Signature*

Secara garis besar, proses verifikasi XML *Signature* adalah sebagai berikut :

a. *Canonicalize* elemen `<SignedInfo>` dengan menggunakan metode pada `<CanonicalizationMethod>`. Hitung ulang nilai digest untuk setiap dokuen yang terdapat dalam elemen `<SignedInfo>` dan bandingkan dengan nilai *digest* yang terdapat pada elemen `<DigestValue>` di dalam elemen `<Reference>` yang bersesuaian.

b. Verifikasi *signature* pada elemen `<SignedInfo>`. Untuk melakukan hal tersebut, hitung ulang nilai *digest* yang dispesifikasikan pada elemen `<SignedInfo>` (dengan menggunakan algoritma *digest* yang dispesifikasikan pada elemen `<SignatureMethod>`) dan gunakan kunci publik yang disediakan untuk melakukan verifikasi bahwa nilai elemen `<SignatureValue>` bersesuaian dengan nilai *digest* pada elemen `<SignedInfo>`.

Perhatikan bahwa proses *canonicalization* yang digunakan tidak akan menimbulkan efek yang tidak diinginkan seperti penulisan ulang **URI**.

5. Sintaks XML *Signature*

Seperti telah dijelaskan sebelumnya, struktur XML *Signature* secara umum dapat dilihat pada kode **XML 1**. Pada bagian ini akan dijelaskan secara lebih mendalam sintaks tiap elemen dari *signature*. Sintaks akan dijelaskan melalui XML *Signature* dan DTD.

5.1. Elemen Signature

Elemen `Signature` elemen dasar (*root*) dari sebuah XML *Signature*. Dalam pengimplementasiannya harus dihasilkan skema XML elemen `Signature` yang valid seperti yang dispesifikasikan dalam skema dibawah ini :

Skema XML 1. Elemen `Signature`

```
Schema Definition:

   <element name="Signature" type="ds:SignatureType"/>
   <complexType name="SignatureType">
     <sequence>
       <element ref="ds:SignedInfo"/>
       <element ref="ds:SignatureValue"/>
       <element ref="ds:KeyInfo" minOccurs="0"/>
       <element ref="ds:Object" minOccurs="0" maxOccurs="unbounded"/>
     </sequence>
```

```
      <attribute name="Id" type="ID" use="optional"/>
   </complexType>

   DTD:

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

5.2. Elemen `SignatureValue`

Elemen `SignatureValue` mengandung nilai aktual dari *digital signature*. Elemen ini selalu di-*encode* dengan menggunakan base64. Skema elemen `SignatureValue` dapat dilihat pada skema dibawah ini:

Skema XML 2. Elemen `SignatureValue`

```
   Schema Definition:

   <element name="SignatureValue" type="ds:SignatureValueType"/>
   <complexType name="SignatureValueType">
     <simpleContent>
       <extension base="base64Binary">
         <attribute name="Id" type="ID" use="optional"/>
       </extension>
     </simpleContent>
   </complexType>

   DTD:

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

5.3. Elemen `SignedInfo`

Struktur elemen `SignedInfo` mencakup antara lain algoritma *canonicalization*, sebuah algoritma *signature*, dan atau atau lebih referensi. Elemen dapat mengandung atribut ID opsional sehingga dapat direferensi oleh *signature* atau objek lain. `SignedInfo` tidak mencakup *signature* eksplisit atau properti *digest* (seperti waktu kalkulasi, nomor serial *device* kriptografi, dan lain-lain). Jika sebuah aplikasi mengasosiasikan propertinya dengan *signature* atau *digest*, maka aplikasi tersebut dapat mencakup informasi tersebut dalam sebuah elemen `SignatureProperties` dalam sebuah elemen `Object`. Skema elemen `SignedInfo` dapat dilihat pada skema dibawah ini :

Skema XML 3. Elemen `SignInfo`

```
   Schema Definition:

   <element name="SignedInfo" type="ds:SignedInfoType"/>
   <complexType name="SignedInfoType">
     <sequence>
       <element ref="ds:CanonicalizationMethod"/>
       <element ref="ds:SignatureMethod"/>
       <element ref="ds:Reference" maxOccurs="unbounded"/>
     </sequence>
     <attribute name="Id" type="ID" use="optional"/>
   </complexType>
```

```
DTD:

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

5.3.1. Elemen CanonicalizationMethod

`CanonicalizationMethod` merupakan elemen yang dibutuhkan untuk menspesifikasikan algoritma yang digunakan dalam proses *canonicalization* yang digunakan pada elemen `SignedInfo` sebelum melakukan operasi penandatanganan. Elemen ini menggunakan struktur umum algoritma yang nanti akan dijelaskan pada upa bab 7 (**Algoritma**).

Aplikasi *signature* harus memperhatikan penerimaan dan pengeksekusian sebuah `CanonicalizationMethod`. Contohnya, `CanonicalizationMethod` harus dapat me-*rewrite* URI pada `Reference` yang sedang divalidasi. Atau, metode ini harus dapat mengubah `SignedInfo` secara keseluruhan sehingga validasi dapat berjalan dengan sukses. Karena `CanonicalizationMethod` berada di dalam `SignedInfo` dan akibat dari bentuk *canonical*, maka elemen ini dapat menghapus dirinya sendiri dari `SignedInfo` atau mengubah elemen SignedInfo sehingga dapat muncul dengan fungsi *canonicalization* yang berbeda. Skema elemen `CanonicalizationMethod` dapat dilihat pada skema dibawah ini :

Skema XML 4. Elemen CanonicalizationMethod

```
Schema Definition:
<element name="CanonicalizationMethod"
type="ds:CanonicalizationMethodType"/>
<complexType name="CanonicalizationMethodType" mixed="true">
<sequence>
<any namespace="##any" minOccurs="0" maxOccurs="unbounded"/>
<!-- (0,unbounded) elements from (1,1) namespace -->
</sequence>
<attribute name="Algorithm" type="anyURI" use="required"/>
</complexType>

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

5.3.2. Elemen `SignatureMethod`

`SignatureMethod` merupakan elemen yang dibutuhkan untuk menspesifikasikan algoritma yang digunakan untuk menghasilkan dan memvalidasi *signature*. Algoritma ini mengidentifikasi semua fungsi kriptografi termasuk dalam operasi penandatanganan (contoh: *hashing*, algoritma kunci publik, *MAC*, *padding*, dan lainnya). Elemen ini menggunakan struktur umum untuk algoritma yang dijelaskan pada upa bab 7 (**Algoritma**). Karena terdapat *identifier* tunggal, *identifier* tersebut dapat menspesifikasikan sebuah format yang mengandung nilai *signature* yang berbeda.

Skema elemen `SignatureMethod` dapat dilihat pada skema dibawah ini :

Skema XML 5. Elemen `SignatureMethod`

```
Schema Definition:
<element name="SignatureMethod" type="ds:SignatureMethodType"/>
<complexType name="SignatureMethodType" mixed="true">
<sequence>
```

```
<element name="HMACOutputLength" minOccurs="0"
type="ds:HMACOutputLengthType"/>
<any namespace="##other" minOccurs="0" maxOccurs="unbounded"/>
<!-- (0,unbounded) elements from (1,1) external namespace -->
</sequence>
<attribute name="Algorithm" type="anyURI" use="required"/>
</complexType>

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

5.3.3. Elemen `Reference`

Elemen `Reference` merupakan elemen yang dapat muncul lebih dari satu kali. Elemen ini menspesifikasikan algoritma *digest* dan nilai *digest* serta (opsional) *identifier* dari objek yang ditandatangan, tipe tanda tangan, dan *transform* mendeskripsikan bagaimana konten yang telah di-*digest* dibuat.

Identifikasi **URI** dan transformasi menjelaskan bagaimana konten yang di-*digest* dibuat. Skema elemen `Reference` dapat dilihat pada skema dibawah ini :

Skema XML 6. Elemen `Reference`

```
Schema Definition:
<element name="Reference" type="ds:ReferenceType"/>
<complexType name="ReferenceType">
<sequence>
<element ref="ds:Transforms" minOccurs="0"/>
<element ref="ds:DigestMethod"/>
<element ref="ds:DigestValue"/>
</sequence>
<attribute name="Id" type="ID" use="optional"/>
<attribute name="URI" type="anyURI" use="optional"/>
<attribute name="Type" type="anyURI" use="optional"/>
</complexType>

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

5.3.3.1 Atribut `URI`

Atribut `URI` mengidentifikasi sebuah data objek menggunakan URI-*Reference*. Karater-karakter yang valid dalam atribut URI sama dengan atribut pada XML. Tipe atribut opsional mengandung informasi tentang tipe objek yang di-*signed*. Contoh :

Kode XML 8. Atribut `URI`

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

Tipe atribut diaplikasikan pada *item* yang sedang ditunjuk, bukan terhadap kontennya. Contohnya, sebuah referensi yang mengidentikasi sebuah elemen Object dan mengandung sebuah elemen `SignatureProperties` adalah tipe `#Object`.

5.3.3.2 Elemen `Transform`

Elemen `Transform` mengandung *list* terurut yang mendeskripsikan bagaiman penandatanganan memperoleh objek data yang di-*digest*. Keluaran dari setiap `Transform` akan menjadi masukan dari `Transform` dari berikutnya. Masukan dari `Transform` pertama berupa hasil deferensi dari atribut **URI** dari elemen Reference. Keluaran dari `Transform` terakhir adalah masukan bagi algoritma DigestMethod. Ketika transformasi diaplikasikan, *signer* tidak ditandai dengan dokumen yang asli namun ditandai dengan dokumen yang ditransformsi.

Setiap `Transform` memiliki sebuah atribut `Algorithm` dan parameter konten yang sesuai dengan algoritma yang diberikan. Nilai atribut `Algorithm` menspesifikasikan nama dari algoritma yang dipakai, dan konten `Transform` menyediakan data tambahan untuk menginstruksikan proses algoritma pada masukan transformasi. Skema elemen ini dapat dilihat pada skema dibawah ini :

Skema XML 6. Elemen `Transform`

```
Schema Definition:
<element name="Transforms" type="ds:TransformsType"/>
<complexType name="TransformsType">
<sequence>
<element ref="ds:Transform" maxOccurs="unbounded"/>
</sequence>
</complexType>
<element name="Transform" type="ds:TransformType"/>
<complexType name="TransformType" mixed="true">
<choice minOccurs="0" maxOccurs="unbounded">
<any namespace="##other" processContents="lax"/>
<!-- (1,1) elements from (0,unbounded) namespaces -->
<element name="XPath" type="string"/>
</choice>
<attribute name="Algorithm" type="anyURI" use="required"/>
</complexType>

DTD:
<!ELEMENT Transforms (Transform+)>
<!ELEMENT Transform (#PCDATA|XPath %Transform.ANY;)* >
<!ATTLIST Transform Algorithm CDATA #REQUIRED >
<!ELEMENT XPath (#PCDATA) >
```

5.3.3.3 Elemen `DigestMethod`

`DigestMethod` merupakan elemen yang dibutuhkan untuk mengidentifikasi algortima *digest* yang akan diterapkan ke objek yang di-*signed*. Skema elemen ini dapat dilihat pada skema di bawah ini :

Skema XML 7. Elemen `DigestMethod`

```
Schema Definition:
<element name="DigestMethod" type="ds:DigestMethodType"/>
<complexType name="DigestMethodType" mixed="true">
<sequence>
<any namespace="##other" processContents="lax" minOccurs="0"
maxOccurs="unbounded"/>
</sequence>
<attribute name="Algorithm" type="anyURI" use="required"/>
</complexType>

DTD:
```

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

5.3.3.4 Elemen `DigestValue`

`DigestValue` merupakan elemen yang mengandung nilai *digest* yang di-*encode*. *Digest* selalu di-*encode* dengan menggunakan *base64*. Contoh skema dapat dilihat di bawah ini :

Skema XML 8. Elemen `DigestValue`

```
Schema Definition:
<element name="DigestValue" type="ds:DigestValueType"/>
<simpleType name="DigestValueType">
<restriction base="base64Binary"/>
</simpleType>
DTD:
<!ELEMENT DigestValue (#PCDATA) >
<!-- base64 encoded digest value -->
```

5.4. Element `KeyInfo`

`KeyInfo` merupakan elemen opsional yang memungkinkan penerima untuk memperoleh kunci yang dibutuhkan untuk memvalidasi *signature*. `KeyInfo` mungkin mengandung kunci, nama, sertifikat dan informasi manajemen kunci publik lainnya, seperti dalam distribusi kunci *band* atau data persetujuan kunci. Spesifikasi ini mendefinisikan sebuah tipe aplikasi yang simpel, tapi aplikasi dapat memperluas tipe tersebut atau bersama-sama menggantikan mereka dengan identifikasi kuncinya sendiri dan saling mengganti semantik menggunakan fasilitas *namespace* XML.

Jika `KeyInfo` dihilangkan, penerima berharap dapat mengidentifikasi kunci berdasarkan konteks aplikasi. Berbagai deklarasi di dalam `KeyInfo` mengacu pada kunci yang sama. Ketika aplikasi-aplikasi dapat mendefinisikan dan menggunakan beberapa mekanisme yang dipilih melalui inklusi pada elemen-elemen dari suatu *namespace* yang berbeda, XML harus mengimplementasikan `KeyValue` dan `RetrievalMethod`.

Berikut ini adalah kumpulan tipe-tipe `KeyInfo` yang dialokasi sebuah *identifier* dalam `&dsig;` *namespace*. Tipe-tipe ini dapat digunakan di dalam atribut `RetievalMethod` Type untuk menguraikan sebuah struktur *remote* `KeyInfo`.

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

Berikut ini adalah contoh skema `KeyInfo` :

Skema XML 9. Elemen `KeyInfo`

```
Schema Definition:
<element name="KeyInfo" type="ds:KeyInfoType"/>
<complexType name="KeyInfoType" mixed="true">
<choice maxOccurs="unbounded">
<element ref="ds:KeyName"/>
<element ref="ds:KeyValue"/>
<element ref="ds:RetrievalMethod"/>
<element ref="ds:X509Data"/>
<element ref="ds:PGPData"/>
<element ref="ds:SPKIData"/>
```

```
<element ref="ds:MgmtData"/>
<any processContents="lax" namespace="##other"/>
<!-- (1,1) elements from (0,unbounded) namespaces -->
</choice>
<attribute name="Id" type="ID" use="optional"/>
</complexType>
DTD:
<!ELEMENT KeyInfo (#PCDATA|KeyName|KeyValue|RetrievalMethod|
X509Data|PGPData|SPKIData|MgmtData %KeyInfo.ANY;)* >
<!ATTLIST KeyInfo
Id ID #IMPLIED >
```

5.4.1. Elemen KeyName

Elemen `KeyName` mengandung sebuah nilai string yang dapat digunakan oleh *signer* untuk memberitahukan sebuah kunci *identifier* kepada penerima. `KeyName` mengandung sebuah *identifier* yang dihubungkan dengan pasangan kunci yang digunakan untuk menandakan pesan, tapi juga dapat mengandung protokol lainnya yaitu informasi terkait yang secara tidak langsung mengidentifikasi sepasang kunci. Contoh skema elemen `KeyName` :

Skema XML 10. Elemen KeyName

```
 Schema Definition:
<element name="KeyName" type="string"/>

DTD:

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

5.4.2. Elemen `KeyValue`

Elemen `KeyValue` mengandung sebuah kunci publik tunggal yang mungkin berguna dalam validasi *signature*. Format yan terstruktur dalam pendefinisian kunci-kunci publik DSA dan RSA didefinisikan dalam **Algoritma-algoritma Signature** yang akan dibahas nanti. Elemen `KeyValue` dapat menyertakan nilai kunci publik eksternal yang direpresentasikan sebagai `PCDATA` atau tipe-tipe elemen dari sebuah *namespace* eksternal. Contoh skema elemen `KeyValue` :

Skema XML 11. Elemen `KeyValue`

```
Schema Definition:

   <element name="KeyValue" type="ds:KeyValueType"/>
   <complexType name="KeyValueType" mixed="true">
    <choice>
      <element ref="ds:DSAKeyValue"/>
      <element ref="ds:RSAKeyValue"/>
      <any namespace="##other" processContents="lax"/>
    </choice>
   </complexType>
   DTD:

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

5.4.2.1. Elemen `DSAKeyValue`

Identifier

`Type="http://www.w3.org/2000/09/xmldsig#DSAKeyValue"`

Contoh skema elemen `DSAKeyValue` :

Skema XML 12. Elemen `DSAKeyValue`

```
Schema Definition:

   <element name="DSAKeyValue" type="ds:DSAKeyValueType"/>
   <complexType name="DSAKeyValueType">
     <sequence>
       <sequence minOccurs="0">
         <element name="P" type="ds:CryptoBinary"/>
         <element name="Q" type="ds:CryptoBinary"/>
       </sequence>
       <element name="G" type="ds:CryptoBinary" minOccurs="0"/>
       <element name="Y" type="ds:CryptoBinary"/>
       <element name="J" type="ds:CryptoBinary" minOccurs="0"/>
       <sequence minOccurs="0">
         <element name="Seed" type="ds:CryptoBinary"/>
         <element name="PgenCounter" type="ds:CryptoBinary"/>
       </sequence>
     </sequence>
   </complexType>

DTD Definition:

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

5.4.2.2. Elemen `RSAKeyValue`

Identifier
Type="http://www.w3.org/2000/09/xmldsig#RSAKeyValue"

`RSAKeyValue` mempunyai dua *field*, yaitu Modulus dan Exponent. Contoh kode :

Kode XML 9. *Field* `RSAKeyValue`

```
 <RSAKeyValue>
   <Modulus>xA7SEU+e0yQH5rm9kbCDN9o3aPIo7HbP7tX6WOocLZAtNfyxSZDU16ksL6W
    jubafOqNEpcwR3RdFsT7bCqnXPBe5ELh5u4VEy19MzxkXRgrMvavzyBpVRgBUwUlV
    5foK5hhmbktQhyNdy/6LpQRhDUDsTvK+g9Ucj47es9AQJ3U=
   </Modulus>
   <Exponent>AQAB</Exponent>
 </RSAKeyValue>
```

Contoh skema elemen `RSAKeyValue` :

Skema XML 12. Elemen `RSAKeyValue`

```
Schema Definition:

   <element name="RSAKeyValue" type="ds:RSAKeyValueType"/>
   <complexType name="RSAKeyValueType">
     <sequence>
       <element name="Modulus" type="ds:CryptoBinary"/>
       <element name="Exponent" type="ds:CryptoBinary"/>
```

```
    </sequence>
  </complexType>
  DTD Definition:

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

5.4.3. Elemen `RetrievalMethod`

Sebuah elemen `RetrievalMethod` di dalam `KeyInfo` digunakan untuk menyampaikan sebuah referensi kepada informasi `KeyInfo` yang disimpan di lokai yang lain. Contoh, beberapa *signature* di dalam sebuah dokumen dapat menggunakan sebuah kunci yang diverifikasi oleh sebuah sertifikat beraintai X.509v3 yang muncul sekali dalam dokumen atau secara *remote* dari luar dokumen. Setiap `KeyInfo` *signature* dapat mereferensikan sertifikat ini dengan menggunkan sebuah elemen `RetrievalMethod` tunggal sebagai ganti dari keseluruhan sertifikat dengan urutan pada elemen-elemen `X509Certificate`. Contoh skema elemen `RetrievalMethod` :

Skema XML 13. Elemen `RetrievalMethod`

```
Schema Definition

  <element name="RetrievalMethod" type="ds:RetrievalMethodType"/>
  <complexType name="RetrievalMethodType">
    <sequence>
      <element ref="ds:Transforms" minOccurs="0"/>
    </sequence>
    <attribute name="URI" type="anyURI"/>
    <attribute name="Type" type="anyURI" use="optional"/>
  </complexType>
DTD

  <!ELEMENT RetrievalMethod (Transforms?) >
  <!ATTLIST RetrievalMethod
     URI   CDATA #REQUIRED
     Type  CDATA #IMPLIED >
```

5.4.4. Elemen `X509Data`

Identifier
`Type="`http://www.w3.org/2000/09/xmldsig#X509Data`"`

Sebuah `X509Data` element di dalam `KeyInfo` mengandung satu atau lebih *identifier* pada kunci-kunci atau sertifikat X509. Konten pada `X509Data` adalah paling tidak satu elemen, dari kumpulan tipe-tipe elemen, dapat muncul bersama-sama atau lebih dari sekali jika dan hanya jika tiap *instance* menguraikan atau dikaitkan pada sertifikat yang sama :

- Elemen `X509IssuerSerial`, yang mengandung sebuah X.509 mencirikan pasangan nomor serial yang memenuhi RFC2253.
- Elemen `X509SubjectName`, yang mengandung sebuah subjek X.509 mencirikan nama yang memenuhi RFC2253.
- Elemen `X509Certificate`, yang mengandung sebuah sertifikat *base64-encoded*.
- Elemen-elemen dari *namespace* eksternal yang menemani beberapa elemen diatasnya.
- Elemen `X509CRL`, yang mengandung sebuah *base64-encoded certificate revolution list* (CRL).

Contoh skema elemen `X509Data` :

Skema XML 14. Elemen `X509Data`

```
Schema Definition
```

```
   <element name="X509Data" type="ds:X509DataType"/>
   <complexType name="X509DataType">
     <sequence maxOccurs="unbounded">
       <choice>
         <elementname="X509IssuerSerial"
         type="ds:X509IssuerSerialType"/>
         <element name="X509SKI" type="base64Binary"/>
         <element name="X509SubjectName" type="string"/>
         <element name="X509Certificate" type="base64Binary"/>
         <element name="X509CRL" type="base64Binary"/>
         <any namespace="##other" processContents="lax"/>
       </choice>
     </sequence>
   </complexType>

   <complexType name="X509IssuerSerialType">
     <sequence>
       <element name="X509IssuerName" type="string"/>
       <element name="X509SerialNumber" type="integer"/>
     </sequence>
   </complexType>

DTD

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

   <!-- Note, this DTD and schema permit X509Data to be empty; this is
   precluded by the text in KeyInfo Element (section 4.4) which states
   that at least one element from the dsig namespace should be present
   in the PGP, SPKI, and X509 structures. This is easily expressed for
   the other key types, but not for X509Data because of its rich
   structure. -->
```

5.5. Elemen `Object`

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

`Object` merupakan elemen opsional yang dapat muncul satu atau lebih. Elemen ini mungkin mengandung data apapun. Elemen `Object` ini dapat mengandung atribut tipe MIME, ID, dan *encoding*.

Atribut `Object'S Encoding` mungkin digunakan untuk menyediakan URI yang mengidentifikasi metode untuk meng-*encode* objek tersebut. Atribut `MimeType` merupakan atribut data yang mendeskripsikan data yang terdapat pada `Object`. Atribut ini berisi nilai *string* yang didefinisikan oleh MIME.

Atribut `Id` umumnya direferensi dari `Reference` di `SignedInfo`. Elemen ini umumnya digunakan pada *enveloping signature* dimana objek yang ditanda tangan dimasukkan ke dalam elemen *signature*. Contoh skema elemen `Object` :

Skema XML 15. Elemen `Object`

```
Schema Definition:

   <element name="Object" type="ds:ObjectType"/>
   <complexType name="ObjectType" mixed="true">
     <sequence minOccurs="0" maxOccurs="unbounded">
       <any namespace="##any" processContents="lax"/>
     </sequence>
     <attribute name="Id" type="ID" use="optional"/>
     <attribute name="MimeType" type="string" use="optional"/>
     <attribute name="Encoding" type="anyURI" use="optional"/>
   </complexType>
   DTD:

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

6. Canonical `XML`

Dokumen XML memiliki sintaks yang tidak terlalu mengikat jika dibandingkan dengan *format* data pada basis data. Hal ini memungkinkan dokumen-dokumen XML, yang ekuivalen dalam konteks aplikasi tertentu, memiliki banyak variasi leksikal dan representasi fisik yang berbeda. Perbedaan tersebut antara lain dalam :

- Struktur entitas dokumen
- Urutan atribut suatu elemen
- *Character encoding*
- Jumlah spasi antara nama elemen dan atribut-atributnya

Sebagai contoh, bandingkan dokumen XML pada kode XML 10 dan kode XML 11 berikut :

Kode XML 10. Dokumen XML

```
<doc>
<a a1=”1” a2=”2”> 123 </a>
</doc>
```

Kode XML 11. Dokumen XML

```
<?xml version=”1.0”
encoding=”UTF-8”?>
<doc>
<a
a2=”2” a1=”1”
>123</a>
<doc>
```

Kedua dokumen tersebut memiliki informasi yang sama, yang ada pada dalam elemen `<a>`. Akan tetapi, dokumen pada Listing 2 memiliki deklarasi XML (`<?xml version=”1.0” encoding=”UTF- 8”?>`) yang tidak terdapat pada Kode XML 10. Selain itu, keduanya memiliki perbedaan jumlah spasi, format penulisan, dan urutan atribut pada elemen `<a>`.

Fleksibilitas leksikal ini menyebabkan beberapa masalah, antara lain dalam *regression testing*, *byte-by-byte comparison*, dan *digital signatures*. Misalnya, Kode XML 10 dikirim melalui sistem pengiriman yang menyebabkan Kode XML 10 berubah menyerupai Kode XML 11. Akibatnya, hash sederhana atau *digital signature* dari kedua listing tersebut tidak akan sama. Padahal, kedua dokumen tersebut memiliki informasi yang sama.

W3C mendefinisikan ***canonical XML*** yang merupakan bentuk leksikal normal untuk dokumen XML. *Canonical XML* ini menghilangkan semua variasi yang diizinkan dan menerapkan aturan yang ketat untuk menghasilkan perbandingan *byte-by-byte* yang konsisten. Proses pembentukannya disebut dengan ***canonicalization*** (populer dengan singkatan “c14n”).

Berdasarkan spesifikasi yang diberikan oleh W3C, pembentukan dokumen XML *canonic* memiliki aturan sebagai berikut :

- Dokumen di-*encode* dalam UTF-8.

- *Line break* dinormalisasikan menjadi "#xA" pada input sebelum diparsing.
- Nilai atribut dinormalisasikan, seolaholah dengan *validating processor*.
- Karakter dan *parsed entity references* diganti.
- Bagian CDATA diganti dengan isi karakternya.
- Deklarasi XML dan *Document Type Declaration* (DTD) dihilangkan karena dokumen sudah dalam UTF-8 sehingga konversi tidak diperlukan lagi.
- Elemen-elemen kosong dikonversi menjadi pasangan *start-end tag*.
- *Whitespace* (spasi, tab, enter) diluar elemen dokumen dan antara *start* dan *end-tags* dinormalisasikan.
- Penanda nilai atribut diubah menjadi *quotation marks* (*double quotes*).
- Karakter-karakter khusus dalam nilai atribut dan isi dokumen diganti dengan referensi karakter.
- Deklarasi *superfluous namespace* dihilangkan dari setiap elemen.
- Atribut *default* ditambahkan ke setiap elemen.
- Urutan leksikografi disesuaikan dengan deklarasi *namespace* dan atribut dari setiap elemen.

Dengan menerapkan aturan *canonicalization*, struktur XML pada Kode XML 10 dan Kode XML 11 akan menjadi seperti pada Kode XML 12. Struktur yang baru ini membuat Kode XML 10 dan Kode XML 11 memiliki perbandingan *byte*-by-*byte* dan nilai hash yang sama.

Kode XML 12.

```
<doc>
<a a1="1" a2="2">123</a>
</doc>
```

7. Algoritma

Bagian ini mengidentifikasi algoritma-algoritma yang digunakan dengan spesifikasi *XML digital signature*. Algoritma yang digunakan dalam *XML Digital Signature* diidentifikasi oleh URI yang muncul sebagai atribut elemen. Algoritma tersebut memiliki parameter yang eksplisit maupun implisit. Parameter eksplisit muncul sebagai elemen isi dalam elemen *algorithm role* yang bersangkutan. Parameter ini memiliki nama elemen deskriptif yang biasanya spesifik terhadap algoritma tertentu Sedangkan paramater implisit tidak ditulis oleh algoritma yang memilikinya, misalnya `SignatureMethod` yang secara impisit mempunyai dua parameter, yaitu *keying info* dan keluaran yang diperoleh dari `CanonicalizationMethod`.

Beberapa algoritma yang dispesifikasikan dalam implementasi antara lain :

a. Digest

Hanya satu algoritm *digest* yang dipakai di sini, yaitu SHA-1. Namun tidak tertutup kemungkinan pemakaian satu atau lebih algoritma baru yang lebih kuat yang dikembangkan oleh **Advanced Encryption Standard (AES)**. Penggunaan **MD5** di sini sangat tidak dianjurkan karena berdasarkan perkembangan terakhir, para kriptanalisis telah sangat tahu kelemahannya.

SHA-1
Identifier :
http://www.w3.org/2000/09/xmldsig#sha1

Algoritma SHA-1 tidak memiliki parameter eksplisit. Contoh elemen algoritma *digest* SHA-1 :

Kode XML 13. Elemen `SHA-1 DigestAlg`

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

Algoritma SHA-1 mempunyai 160 bit *string*. Isi elemen `DigestValue` harus berdasarkan *base64 encoding* pada *string* bit yang muncul sebagai 20 oktet *stream*. Contoh elemen `DigestValue` untuk *digest* pesan :

Kode XML 14. Elemen `DigestValue`

```
A 9993E36 4706816A BA3E2571
7850C26C 9CD0D89D
```

Sehingga berdasarkan standar SHA-1 menjadi :

Kode XML 15. Elemen Standar SHA-1

```
<DigestValue>qZk+NkcGgWq6PiVxeFDC
bJzQ2J0=</DigestValue>
```

b. Encoding

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

Dalam XML digital signature, digunakan encoding base64. Identifier yang digunakan adalah :

Kode XML 16. Elemen Elemen *base64*

```
<SignatureMethod
Algorithm=”http://www.w3.org/200
0/09/xmldsig#base64”>
```

c. MAC

Algoritma MAC mempunyai dua parameter implisit. Nilai kunci dideterminasi dari `KeyInfo` dan keluaran *stream* oktet dari `CanonicalizationMethod`. Algoritma MAC dan *signature* identik secara sintaks namun MAC menyiratkan kunci rahasia yang bersifat *shared.*

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

Algoritma HMAC panjang *truncation* dalam bit sebagai parameternya. Jika parameter tidak dispesifikasikan maka semua bit *hash* adalah sebagai keluarannya. Contoh elemen `SignatureMethod` HMAC :

Kode XML 17. Elemen `SignatureMethod` HMAC 1

```
<SignatureMethod
Algorithm="http://www.w3.org/200
0/09/xmldsig#hmac-sha1">

<HMACOutputLength>128</HMACOutput
Length>
   </SignatureMethod>
```

Keluaran dari algoritma HMAC adalah keluaran pada algoritma *digest* yang dipilih. Nilai ini harus merupakan *base64 encoded* dalam penunjukkan langsung yang sama sebagai keluaran algoritma *digest*, Contoh elemen `SignatureValue` untuk *digest* HMAC-SHA1 :

Kode XML 18. Elemen `SignatureMethod` HMAC-SHA2

```
9294727A 3638BB1C 13F48EF8

158BFC9D
```

Sehingga berdasarkan vektor tes dalam HMAC menjadi :

Kode XML 19. Elemen `SignatureMethod` HMAC

```
<SignatureValue>kpRyejY4uxwT9I74F
Yv8nQ==</SignatureValue>
```

Contoh skema HMAC :

Skema XML 16. Skema HMAC

```
Schema Definition:

   <simpleType
name="HMACOutputLengthType">
     <restriction
base="integer"/>
   </simpleType>
   DTD:

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

d. Signature

Algoritma *signature* mempunyai dua parameter implisit. Nilai kunci dideterminasi dari `KeyInfo` dan keluaran *stream* oktet dari `CanonicalizationMethod`. *Signature* dan algoritma MAC identik secara sintaks namun sebuah *signature* menyiratkan kriptografi kunci publik.

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

Algoritma DSA tidak mempunyai parameter eksplisit. Contoh elemen `SignatureMethod` DSA :

Kode XML 20. Elemen `SignatureMethod` DSA

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

Keluaran pada algoritma DSA terdiri dari sepasang *integer*s yang biasanya diacu oleh pasangan. Nilai *signature* terdiri dari *base64 encoding* pada penggabungan dua *stream* oktet yang berturut-turut dihasilkan dari *encoding* oktet dari nilai r dan s dalam urutan tertentu. Contoh elemen `SignatureValue` untuk sebuah *signature* DSA `(r, s)` dengan nilai dalam format heksadesimal :

Kode XML 21. Elemen `SignatureMethod` DSA(r, s)

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

Sehingga berdasarkan standar DSS menjadi :

Kode XML 22. Standar DSS

```
<SignatureValue>
i6watmQQQ1y3GB+VsWq5fJKzQcBB4jRfH
1bfJFj0JtFVtLotttzYyA==</Signatur
eValue>
```

e. Canonicalization

Jika *canonicalization* bekerja dengan format oktet, maka algoritma *canonicalization* mempunyai dua parameter implisit, yaitu konten dan *charset*. *Charset* diturunkan berdasarkan aturan protokol *transport* dan tipe media. Informasi ini dibutuhkan untuk memeriksa tanda tangan dan untuk memverifikasi dokumen dan selalu menginginkan konfigurasi sisi server yang aman.

Berbagai macam algoritma *canonicalization* memerlukan konversi ke UTF-8. Algoritma ini dapat menghilangkan maupun mempertahankan *comment*.

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

Contoh elemen `Canonicalization` XML dengan menghilangkan *comments* :

Kode XML 23. Elemen `Canonicalization`

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

Identifier for REQUIRED Canonical XML with comments :
http://www.w3.org/TR/2001/REC-xml-c14n-20010315#WithComments

Contoh elemen canonicalization XML dengan mempertahankan *comments* :

Kode XML 24. Elemen `Canonicalization with comments`

```
<SignatureMethod
Algorithm=”http://www.w3.001/REC-
xml-c14n-20010315#WithComments”>
```

`f. Transform`

Algoritma `Transform` memiliki sebuah parameter tunggal, yaitu sebuah *stream* oktet dari `Reference` atau dari keluaran `Transform` yang lebih dulu. Untuk transformasi dari dokumen XML menjadi format lain, dapat digunakan beberapa macam algoritma.

Base64
Identifiers :
http://www.w3.org/2000/09/xmldsig#base64

Spesifikasi normatif pada transformasi *base64 decoding* adalah `MIME`. Elemen `Transform` *base64* tidak mempunyai konten. Masukannya di-*decode* oleh algoritma. Transformasi ini berguna jika sebuah aplikasi ingin menandatangani data mentah yang berhubungan dengan konten yang ter-*encode* pada sebuah elemen. Contoh elemen *base64* :

Kode XML 25. Elemen *`base64`*

```
<SignatureMethod
Algorithm="http://www.w3.org/2000
/09/xmldsig#base64">
```

XSLT Transform
Identifier :
http://www.w3.org/TR/1999/REC-xslt-19991116

Tranformasi dengan XSLT (optional) dapat menghasilkan keluaran HTML. Contoh elemen XSLT :

Kode XML 26. Elemen `XSLT`

```
<SignatureMethod
Algorithm="http://www.w3.org/TR/
1999/REC-xslt-19991116">
```

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

Tujuan utama dari XPath *filtering* adalah untuk meyakinkan bahwa hanya perubahan yang didefinisikan secara spesifik terhadap dokumen

XML masukan yang diizinkan setelah *signature* ditambahkan. Contoh elemen XPath :

Kode XML 27. Elemen `XPath`

```
<SignatureMethod
Algorithm="http://www.w3.org/TR/
1999/REC-xpath-19991116">
```

Enveloped Signature Transform Identifier :
http://www.w3.org/2000/09/xmldsig#enveloped-signature

Transformasi ini menghapus seluruh elemen *signature* yang mengandung **T** dari perhitungan *digest* dalam elemen Reference yang mengandung **T**.

Kode XML 28. Elemen `Reference T`

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

8. Implementasi XML *Signature* dalam Microsoft .NET

Pada bab ini akan dibahas mengenai pembangunan perangkat lunak yang bernama **XJS (XML Joga Signature)** berfungsi untuk pengimplementasian XML *Signature* dalam Microsoft .NET *framework* yaitu dengan menggunakan bahasa pemrograman Visual C#. Terdapat tiga *namespace* utama yang diapakai dalam pembangunan XJS, yaitu :

a. **`System.Security.Cryptography`**

Dalam *namespace* ini digunakan *class* `DSA` (*Digital Signature Algorithm*) yang mengandung semua implementasi dari algoritma DSA. Namun *method* yang digunakan hanya `create()` yang berfungsi untuk mencipatakan sebuah objek DSA yang digunakan dalam operasi algoritma asimetris, juga digunakan *method* `ToXmlString()` yang berfungsi untuk merepresentasikan objek DSA dalam bentuk sebuah XML *string*.

b. **`System.Xml`**

Dalam *namspace* ini *classes* yang digunakan antara lain `XMLDocument`, `XMLElement`, dan `XMLNodeList`.

c. **`System.Security.Cryptography.Xml`**

Dalam *namespace* ini *classes* yang digunakan antara lain `DataObject`, `SignedXml`, dan `KeyInfo`.

XJS terdiri dari sebuah jendela utama (**MainForm**) dan tiga jendela lain yang berfungsi untuk membangkitkan kunci, penandatanganan dokumen XML, dan otentikasi dokumen *Signed*XML. Berikut adalah implementasi **MainForm** dalam Visual C# :

Kode Program 1. MainForm.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;

namespace WindowsApplication1
{
    public partial class MainForm : Form
    {

        private String currFile;
        private String fileContent;

        public Form1()
        {
            InitializeComponent();
            currFile = "";
```

```
        }

        public String getFileAsString(String path)
        {
            String result = "";
            try
            {
                using (System.IO.StreamReader sr = new
System.IO.StreamReader(path))
                {
                    result = sr.ReadToEnd(); //Membaca sampai EoF
                }
            }
            catch (Exception e)
            {
                MessageBox.Show("Error! \r \nFile not found");
                Console.Write(e.ToString());
            }
            return result;
        }

        public void writeStringToFile(String text, String path)
        {
            System.IO.StreamWriter sr =
System.IO.File.CreateText(path);
            sr.Write(text);
            sr.Close();
        }

        public Boolean isSignedDoc(String path)
        {
            Boolean found = false;
            if (fileContent.IndexOf("</Signature>") != -1)
            {
                found = true;
            }
            return found;
        }

        public String getCurrFile()
        {
            return this.currFile;
        }

        private void btnOpen_Click(object sender, EventArgs e)
        {
            openFileDialog1.InitialDirectory = "c:\\";
            openFileDialog1.Filter = "XML files (*.xml)|*.xml|All files
(*.*)|*.*";
            openFileDialog1.FilterIndex = 1;
            openFileDialog1.RestoreDirectory = true;
            openFileDialog1.FileName = "";
            if (openFileDialog1.ShowDialog() == DialogResult.OK)
            {
                if (openFileDialog1.OpenFile() != null)
                {
                    fileContent =
```

```
getFileAsString(openFileDialog1.FileName);
                    mainText.Text = fileContent;
                    currFile = openFileDialog1.FileName;
                    textBox1.Text = currFile;
                    btnSign.Enabled = true;
                    if (isSignedDoc(textBox1.Text))
                    {
                        btnAutentication.Enabled = true;
                    }
                }
            }
        }

        private void btnClose_Click(object sender, EventArgs e)
        {
            currFile = "";
            mainText.Text = "";
            textBox1.Text = "";
            fileContent = "";
            btnSign.Enabled = false;
            btnKeyGen.Enabled = true;
            btnAutentication.Enabled = false;
        }

        private void btnSaveAs_Click(object sender, EventArgs e)
        {
            saveFileDialog1.FileName = textBox1.Text;
            if (saveFileDialog1.ShowDialog() == DialogResult.OK)
            {
                writeStringToFile(mainText.Text,
saveFileDialog1.FileName);
                textBox1.Text = saveFileDialog1.FileName;
                currFile = textBox1.Text;
                fileContent = mainText.Text;
            }
        }

        private void btnExit_Click(object sender, EventArgs e)
        {
            Application.Exit();
        }

        private void btnSign_Click(object sender, EventArgs e)
        {
            Sign sign = new Sign(currFile);
            sign.ShowDialog();
        }

        private void btnKeyGen_Click(object sender, EventArgs e)
        {
            KeyGen keyGen = new KeyGen();
            keyGen.ShowDialog();
        }

        private void btnAutentication_Click(object sender, EventArgs e)
        {
            Authentication authentication = new
```

```
Authentication(currFile);
            authentication.ShowDialog();
        }
    }
}
```

Di bawah ini adalah tampilan awal XJS yang menampilkan jendela utama dan pembukaan sebuah file xml :

Gambar 1. Jendela MainForm

Bila tombol **Open** ditekan maka akan muncul sebuah OpenFileDialog, lalu isi suatu file yang dipilih (XML biasa maupun *signed*XML) akan ditampilkan seperti terlihat pada tampilan diatas. Pada tampilan diatas ditampilkan isi dari file CreditcardInfo.xml. Perlu diketahui bahwa tombol **Sign XML File** pada awalnya tidak aktif (*property* `enabled` sama dengan `false`), tombol ini aktif kembali bila telah ada sebuah file yang dibuka. Begitu pula tombol **Authentication** awalnya tidak aktif, dan hanya akan aktif bila file yang dibuka adalah sebuah file XML yang telah disign (*Signed*XML).

Perangkat lunak ini mempunyai tiga fungsi utama, yaitu :

a. Pembangkitan kunci

Pembangkitan kunci dilakukan pada jendela **KeyGen**. Jendela ini muncul bila ditekan tombol **GenerateKey** pada jendela **MainForm**. Pada proses pembangkitan kedua jenis kunci di bawah ini digunakan *method* `create()` dan `ToXmlString()` dalam kelas DSA yang terdapat dalam **`System.Security.Cryptography`** *namespace*. Pada XJS ini akan dibangkitkan dua jenis kunci, yaitu :

i. Kunci gabungan (***.keypair**). Kunci ini merupakan kunci gabungan antara *private key* dan *public key*. Kunci ini sebenarnya merupakan format XML, namun disimpan dalam format ***.keypair**, agar lebih mudah dalam membedakan jenis kunci.

ii. Kunci publik (***.pubjoga**). Kunci ini adalah *public key* yang digunakan sewaktu melakukan verifikasi/otentikasi dokumen nantinya. Kunci ini dibangkitkan bersama-sama dengan kunci gabungan sewaktu pembangkitan kunci. Kunci ini juga sebenarnya merupakan format XML, namun disimpan dalam format ***.pubjoga**, agar

lebih mudah dalam membedakan jenis kunci.

Berikut adalah implementasi modul **KeyGen** :

Kode Program 2. KeyGen.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using System.IO;
using System.Security.Cryptography;
using System.Security.Cryptography.Xml;
using System.Xml;

namespace WindowsApplication1
{
    public partial class KeyGen : Form
    {
        private DSA objDSAkey;
        public KeyGen()
        {
            InitializeComponent();
        }

        public void writeKeyToFile(String path, String key)
        {

            System.IO.StreamWriter sr =
System.IO.File.CreateText(path);
            sr.Write(key);
            sr.Close();
        }

        private void btnGenKeyPair_Click(object sender, EventArgs e)
        {
            objDSAkey = DSA.Create();
            string strDSAKeyXML = objDSAkey.ToXmlString(true);
            KeyTextBox.ReadOnly = true;
            KeyTextBox.Text = strDSAKeyXML;
            btnSaveKey.Enabled = true;
            btnGenPubKey.Enabled = true;
            btnSaveKey.Text = "Save Key Pair To";
            saveFileDialog1.Filter = "Keypair files
(*.keypair)|*.keypair";
        }

        private void btnSaveKey_Click(object sender, EventArgs e)
        {
            if (saveFileDialog1.ShowDialog() == DialogResult.OK)
            {
                string keypair = KeyTextBox.Text;
                writeKeyToFile(saveFileDialog1.FileName, keypair);
            }
        }
```

```
        private void btnGenPubKey_Click(object sender, EventArgs e)
        {

            string strDSAKeyXML = objDSAkey.ToXmlString(false);
            KeyTextBox.ReadOnly = true;
            KeyTextBox.Text = strDSAKeyXML;
            btnSaveKey.Enabled = true;
            btnSaveKey.Text = "Save Public Key To";
            saveFileDialog1.Filter = "PublicKey files
(*.pubjoga)|*.pubjoga";
        }

        private void btnClose_Click(object sender, EventArgs e)
        {
            this.Close();
        }
    }
}
```

Terdapat *instance* `objDSAkey` dari *class* `DSA` yang mengandung semua implementasi dari algoritma DSA, yaitu dengan memanggil *method* `create()` yang berfungsi untuk mencipatakan sebuah objek DSA yang digunakan dalam operasi algoritma asimetris. Lalu digunakan *method* `ToXmlString()` yang berfungsi untuk merepresentasikan objek DSA dalam bentuk sebuah XML *string*. Parameter pada *method* inilah yang kemudian membedakan pembangkitan *keypair* (`btnKeyPair_Click()`) dan *public key* (`GenPubKey_Click()`). Bila isi parameter dalam method bernilai `true` (`ToXmlString(true)`) maka string `strDSAKeyXML` yang dihasilkan adalah *keypair*. Sebaliknya bila bernilai false maka string `strDSAKeyXML` yang dihasilkan adalah *public key* (pubjoga). Kedua jenis kunci ini kemudian disimpan dengan nama kunci sesuai masukan *user*. Berikut adalah tampilan jendela **KeyGen** dan contoh tampilan pembangkitan *keypair* :

Gambar 2. Jendela KeyGen *Keypair*

Tampilan diatas adalah jendela **KeyGen** dan setelah menekan tombol **Generate Key Pair**. Tampilan pada **RichTextBox** adalah nilai kunci *keypair*. Kunci tersebut dapat disimpan dengan menggunakan tombol **Save Key Pair To** dalam bentuk format ***.keypair** (contoh : key.keypair).

Berikut adalah isi dari file **key.keypair** :

Kode Kunci 1. key.keypair

```
<DSAKeyValue><P>
9/PectKug0zolIPca3PF5NjEG5nyU/QwyetMB1Mjg5INb4vVeUiiMKbiX7oYZmi9G3lyUsKmoHavc
hfb8By7RxTm8lByYHSbA42QLU65yYYekQGHzeToCQfFo7CAMLpL5LlvJ2TP9zgnWOUQ484d
uiEEydOiD9Qnb/VVEj6CuPU=</P><Q>7I0V0MFodIuHdOKkpOkQAy8A7vc=</Q><G>
ygSoI+awxY3ANYZ+ZrvL+vpUlrb9u/GXvgrOfhkjK0wbzMrvVwBpo8R5bjdSNP4aA0v9ARcecO3
ym9GUjXwiqMGic+jExtthFP6w8JYCIk8VF5L9IHiWXuJCBfNYZRWIug4nsFsTAzNH1afh1LzgZU
FTYu8h5l11fdBbWkZwtu8=</G><Y>JW8dXINuwbGIBy/+
2pnYdhh0K9E9dUJWupe2e9TRAFldxd8FWu6sCWF3vMT9mBUbY2Vuu5urFbx8pDMBfR8NES
I0ZR3j4CSkTp6sQ5zyWm+HP6sn1SaGH53IAG4syLToeEXtPdd2ZfloZ3F2QIfdR7jzRfoykaj/oxjX
Wvs/Bsg=</Y><J>
AAAAAQxWwwgaB8DwwZbn1EpL4Xy4Dzf2SvUG6kbcm85srHHbvpHp1VeZU805P8NdCJauq
29ZaDtyViZ9SULUXDQHMCMZ5/PxDahylvvEkCkxWhA+m/Sg6/PwbsTaF0paIxib4eisMyfGI5A
SzTVtrA==</J><Seed>sc1vARdtOoZ2SkkAP8NfEjn8H/8=</Seed><PgenCounter>Qg==
</PgenCounter><X>NBrX0CnE5DKkKwm9oaH8YYoclsw=</X></DSAKeyValue>
```

Kunci publik dapat dibangkitkan dengan menekan tombol **Generate Public Key**. Seperti tampilan dibawah ini :

Gambar 3. Jendela KeyGen Kunci Publik

Secara otomatis tombol di bawah berubah teksnya menjadi tombol **Save Public Key To**, kunci publik dapat disimpan (**key.pubjoga**) dalam format ***.pubjoga** dengan menekan tombol tersebut.

Berikut adalah isi dari file **key.pubjoga** :

Kode Kunci 2. key.pubjoga

```
<DSAKeyValue><P>
9/PectKug0zolIPca3PF5NjEG5nyU/QwyetMB1Mjg5INb4vVeUiiMKbiX7oYZmi9G3lyUsKmoHavc
```

```
hfb8By7RxTm8lByYHSbA42QLU65yYYekQGHzeToCQfFo7CAMLpL5LlvJ2TP9zgnWOUQ484d
uiEEydOiD9Qnb/VVEj6CuPU=</P><Q>7I0V0MFodIuHdOKkpOkQAy8A7vc=</Q><G>
ygSoI+awxY3ANYZ+ZrvL+vpUlrb9u/GXvgrOfhkjK0wbzMrvVwBpo8R5bjdSNP4aA0v9ARcecO3
ym9GUjXwiqMGic+jExtthFP6w8JYCIk8VF5L9IHiWXuJCBfNYZRWIug4nsFsTAzNH1afh1LzgZU
FTYu8h5l11fdBbWkZwtu8=</G><Y>JW8dXINuwbGIBy/+
2pnYdhh0K9E9dUJWupe2e9TRAFldxd8FWu6sCWF3vMT9mBUbY2Vuu5urFbx8pDMBfR8NES
I0ZR3j4CSkTp6sQ5zyWm+HP6sn1SaGH53IAG4syLToeEXtPdd2ZfloZ3F2QIfdR7jzRfoykaj/oxjX
Wvs/Bsg=</Y><J>
AAAAAQxWwwgaB8DwwZbn1EpL4Xy4Dzf2SvUG6kbcm85srHHbvpHp1VeZU805P8NdCJauq
29ZaDtyViZ9SULUXDQHMCMZ5/PxDahylvvEkCkxWhA+m/Sg6/PwbsTaF0paIxib4eisMyfGI5A
SzTVtrA==</J><Seed>sc1vARdtOoZ2SkkAP8NfEjn8H/8=</Seed><PgenCounter>Qg==
</PgenCounter></DSAKeyValue>
```

Perhatikan bahwa perbedaan antara *keypair* dan *public key* hanya pada elemen terakhir, pada *public key*, tidak terdapat elemen `<X>` yang merupakan *private key*.

b. Penanda tanganan dokumen XML

Penandatanganan dokumen XML dilakukan pada jendela **Sign**. Jendela ini muncul bila ditekan tombol **Sign XML File** pada jendela **MainForm**. Terdapat beberapa *object* yang diciptakan dari beberapa kelas tambahan dari *namespace* yang telah disebutkan di atas, antara lain :

i. `DataObject`

`DataObject` merupakan objek yang digunakan untuk menyimpan `Data` yang ingin di-*signed*. `DataObject` pada .NET mengacu pada tag XML di dalam spesifikasi W3C (kode 1). `DataObject` mempunyai dua properti, yaitu `Data` dan `ID`. Data berfungsi untuk mengeset atau mendapatkan `Data` yang di-*signed*, sedangkan `ID` berfungsi mereferensi `Data` pada lokasi yang berbeda. Contoh kode berikut akan menciptakan sebuah `DataObject` dan men-set `Data` sesuai dengan nama file yang disimpan dalam "**textBoxFileName.Text"** :

Kode Program 3. `DataObject` *Class*

```
XmlDocument objdocument = new XmlDocument();
objdocument.Load(textBoxFileName.Text);
DataObject dataObject = new DataObject();
dataObject.Data = objdocument.ChildNodes;
dataObject.Id = "CreditCardInfo";
```

Dengan menggunakan properti `ID` pada `DataObject` dapat dilakukan pengacuan pada sebuah file di Internet.

ii. `Reference`

Objek `Reference` dalam .NET *framework* merepresentasikan elemen `<Reference>` pada tag XML sesuai spesifikasi W3C. Bisa terdapat lebih dari satu tag `Reference` dalam sebuah dokumen XML yang di-*signed* (*Signed*XML). `Reference` memberikan fleksibilitas dalam menambahkan berbagai `Data` yang di-*signed* dan menerapkan transformasi yang berbeda bagi setiap `DataObject`. Nilai `DigestValue` spesifik pada sebuah `URI Reference`. Kode berikut akan menciptakan sebuah `Object Reference` dan sekumpulan referensi terhadap `DataObject` yang sebelumnya diciptakan :

Kode Program 4. `Reference` *Class*

```
Reference reference = new Reference();
reference.Uri = "#CreditCardInfo";
```

Class `Reference` dalam .NET *framework* memberikan fleksibilitas dalam penambahan `Transformations`. `Transformations` diterapkan pada dokumen XML (`DataObject`) sebelum `Digest` dihitung.

iii. `SignedXML`

`SignedXML` mempunyai *methods* untuk melakukan perhitungan dan verifikasi terhadap

`SignatureValue`. Untuk meng-*sign* sebuah dokumen XML, elemen yang dibutuhkan adalah `Data` yang di-*signed* dan kunci yang digunakan untuk meng-*sign* `Data`. `SignedXML` mempunyai beberapa *method* yang dipakai, antara lain :

- `SigningKey`, berfungsi untuk meng-*set* kunci yang dipakai untuk meng-*sign* dokumen XML.

- `AddObject`, berfungsi untuk menambahkan `DataObject` (`Data` yang di-*signed*) pada Objek `SignedXML`.

- `AddReference`, berfungsi untuk menambahkan `Reference` pada `DataObject` sepanjang beberapa Information transformasi.

- `ComputeSignature`, berfungsi untuk menghitung XML *Signature* dengan menggunakan informasi yang tersedia (`SigningKey`, `DataObject`, `Reference`).

Kode berikut akan mengambil dari file ***.keypair** sesuai dengan yang ada pada "**teksBoxKeyName.Text**" dan meng-*set* kunci tersebut dengan menggunakan `SigningKey` pada `SignedXML`.

Kode Program 5. Pengambilan Kunci

```
XmlDocument objDSAKeyPairXMLdocument = new XmlDocument();
objDSAKeyPairXMLdocument.Load(textBoxKeyName.Text);
string strDSAKeyPairXML = objDSAKeyPairXMLdocument.InnerXml;
SignedXml signedXml = new XmlSignedXml();
DSA objDSAkeyPair = DSA.Create();
objDSAkeyPair.FromXmlString(strDSAKeyPairXML);
signedXml.SigningKey = objDSAkeyPair;
```

Kode berikut akan menambahkan `DataObject` pada `SignedXML` sesuai yang dihasillkan pada **Kode Program 3** :

Kode Program 6. Penambahan `DataObject`

```
signedXml.AddObject(dataObject);
```

Kode berikut akan menambahkan objek `Reference` pada `SignedXML` sesuai yang dihasilkan pada **Kode Program 4** :

Kode Program 7. Penambahan `Reference`

```
signedXml.AddReference(reference);
```

Kode berikut akan menghitung/membangkitkan informasi *signature* :

Kode Program 8. Pembangkitan *Signature*

```
signedXml.ComputeSignature();
```

Kode akan membuat representasi `SignedXML` dalam bentuk format XML :

Kode Program 9. `SignedXML` dalam format XML

```
XmlElement xmlSignature = signedXml.GetXml();
```

Berikut adalah implementasi modul **Sign** lengkapnya :

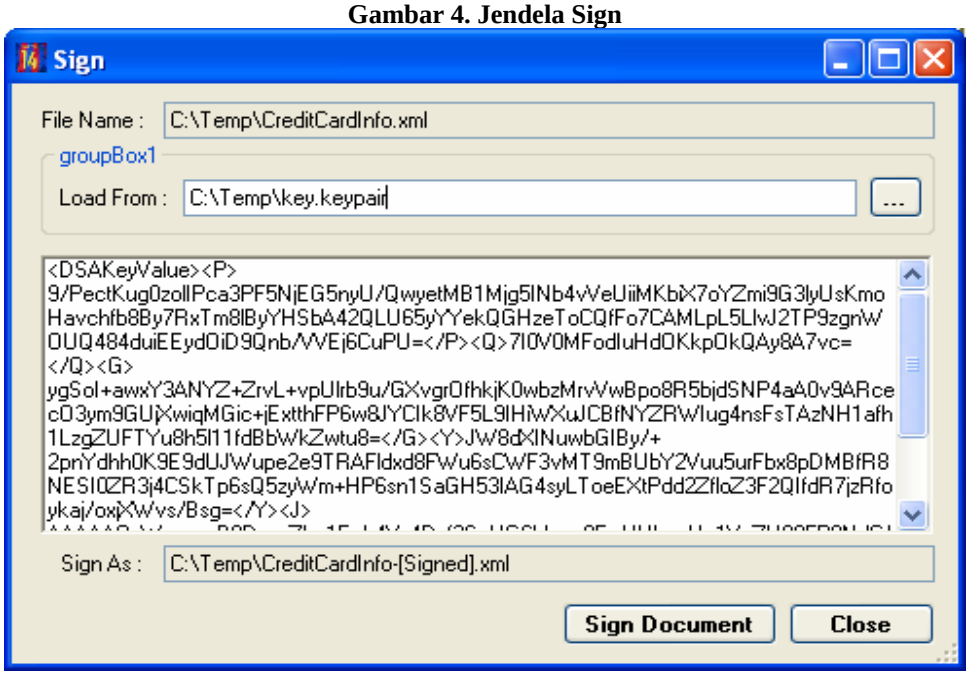
Kode Program 10. Sign.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using System.IO;
using System.Security.Cryptography;
using System.Security.Cryptography.Xml;
using System.Xml;

namespace WindowsApplication1
{
    public partial class Sign : Form
    {
        public Sign()
        {
            InitializeComponent();
        }
        public Sign(String path)
        {
            InitializeComponent();
            textBoxFileName.Text = path;
            int lastIdx = path.LastIndexOf('.');
            textBoxSignFileName.Text = path.Substring(0, lastIdx) + "-
[Signed].xml";
        }

        public String getFileAsString(String path)
        {
            String result = "";
            try
            {
                using (System.IO.StreamReader sr = new
System.IO.StreamReader(path))
                {
                    result = sr.ReadToEnd(); //Membaca sampai EoF
                }
            }
            catch (Exception e)
            {
                MessageBox.Show("Error! \r \nFile not found");
                Console.Write(e.ToString());
            }
            return result;
        }

        public void writeStringToFile(String text, String path)
        {
            System.IO.StreamWriter sr =
System.IO.File.CreateText(path);
            sr.Write(text);
            sr.Close();
        }
```

```
        private void btnLoadKeyPair_Click(object sender, EventArgs e)
        {
            if (openKeyFileDialog1.ShowDialog() == DialogResult.OK)
            {
                string keypair;
                textBoxKeyName.Text = openKeyFileDialog1.FileName;
                keypair = getFileAsString(textBoxKeyName.Text);
                KeyPairText.ReadOnly = true;
                KeyPairText.Text = keypair;
                btnSign.Enabled = true;
            }
        }

        private void btnSign_Click(object sender, EventArgs e)
        {
            XmlDocument objDSAKeyPairXMLdocument = new XmlDocument();
            objDSAKeyPairXMLdocument.Load(textBoxKeyName.Text);
            string strDSAKeyPairXML =
objDSAKeyPairXMLdocument.InnerXml;

            SignedXml signedXml = new XmlSignedXml();
            DSA objDSAkeyPair = DSA.Create();
            objDSAkeyPair.FromXmlString(strDSAKeyPairXML);
            signedXml.SigningKey = objDSAkeyPair;

            XmlDocument objdocument = new XmlDocument();
            objdocument.Load(textBoxFileName.Text);

            DataObject dataObject = new DataObject();
            dataObject.Data = objdocument.ChildNodes;
            dataObject.Id = "CreditCardInfo";

            signedXml.AddObject(dataObject);

            Reference reference = new Reference();
            reference.Uri = "#CreditCardInfo";

            signedXml.AddReference(reference);
            signedXml.ComputeSignature();

            XmlElement xmlSignature = signedXml.GetXml();
            writeStringToFile(xmlSignature.OuterXml,
textBoxSignFileName.Text);
            this.Close();
            MessageBox.Show("XML File has been signed");
        }

        private void btnClose_Click(object sender, EventArgs e)
        {
            this.Close();
        }
    }
}
```

Berikut adalah tampilan jendela **Sign** dengan menggunakan file CreditcardInfo.xml sesuai pada **Gambar 1** dan menggunakan kunci **key.keypair** sesuai dengan **Kode Kunci 1**.

Gambar 4. Jendela Sign



Kunci dalam **key.keypair** dapat di-*load* dengan menekan tombol "**...**", sehingga muncul nilai kunci pada **KeyPairText**. Untuk melakukan *sign* dokumen dapat menekan tombol **Sign Document** yang otomatis akan menyimpan *signed* XML pada direktori yang sama dengan dokumen XML awal, yaitu dengan menambahkan "**-[Signed]**" pada nama file (dalam hal ini CreditcardInfo-[Signed].xml).

Berikut adalah keluaran CreditcardInfo-[Signed].xml, bandingkan dengan spesifikasi W3C pada **Kode XML 2** :

Isi File 1. Sign Document : CreditcardInfo-[Signed].xml

```
<Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
  <SignedInfo>
  <CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20
   010315" />
  <SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#dsa-sha1" />
  <Reference URI="#CreditCardInfo">
    <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
    <DigestValue>MCJRwFzUQ+W6YQer9tiULOKEAUU=</DigestValue>
  </Reference>
  </SignedInfo>
<SignatureValue>yex/wl3NP3/hwtmlpRVRyL30vHUHM1o4BGTOt/h71s2GY69iBg6eZA==
</SignatureValue>
  <Object Id="CreditCardInfo">
    <CreditCardInfo xmlns="">
      <CCNumber>13503046</CCNumber>
      <Amount>100</Amount>
      <FirstName>Diko</FirstName>
      <LastName>Aldillah</LastName>
      <Address1>Cisitu Lama No.10/160B</Address1>
      <City>Bandung</City>
      <Zip>14120</Zip>
     </CreditCardInfo>
    </Object>
```

```
</Signature>
```

c. Otentikasi dokumen *Signed*XML

Otentikasi dokumen XML yang telah di-*sign* (*Signed*XML) dilakukan pada jendela **Authentication**. Jendela ini akan muncul bila ditekan toombol **Authentication** pada **MainForm**. Otentikasi dilakukan hanya terhadap dokumen XML yang telah di-*signed*, oleh karena itu tombol **Authentication** hanya akan aktif bila dokumen yang dibuka mengandung elemen `</Signature>`. Otentikasi adalah mengecek validitas dari dokumen *Signed*XML, untuk itu `KeyValue` harus diketahui dalam hal ini kunci publik dari dokumen *Signed*XML tersebut. `KeyInfo` tidak dikirim sebagai salah satu bagian dari dokumen asli. Hal ini bersifat opsional dan bergantung pada kebijakan pengembang aplikasi untuk menyertakan `KeyInfo` atau tidak. Namun untuk alasan keamanan, sangat direkomendasikan untuk memisahkan kunci publik.

Terdapat kelas lain yang dipakai dari *namespace* **`System.Security.Cryptography.Xml`**, yaitu `KeyInfo`. Kelas ini memberikan fleksibilitas untuk menambahkan beberapa `KeyValue` seperti RSA, DSA, PGP, dan lainnya dengan menggunakan *method* `AddClause`. Berikut adalah kode sebelum melakukan `CheckSignature` :

Kode Program 11. Penambahan `KeyValue`

```
KeyInfo keyInfo = new KeyInfo();
keyInfo.AddClause(new DSAKeyValue(objDSAkeyPair));
signedXml.KeyInfo = keyInfo;
```

Ketika penerima menerima dokumen *Signed*XML, dia menggunakan kunci publik untuk mendekripsi data sehingga mendapatkan `MessageDigest`. Aplikasi lalu membuat `MessageDigest` dan membandingkan apakah data telah mengalami modifikasi atau tidak.

Berikut adalah kode untuk me-*load* file yang berisi kunci publik dan mengubahnya ke *string* `signedxmldata` dengan menggunakan *method* `getFileAsString` :

Kode Program 12. *Load public key* dan ubah ke *string*

```
XmlDocument objDSAKeyPairXMLdocument = new XmlDocument();
objDSAKeyPairXMLdocument.Load(PublicKeyTextBox.Text);
string strDSAKeyPairXML = objDSAKeyPairXMLdocument.InnerXml;
String signedxmldata = getFileAsString(FileNameTextBox.Text);
```

Kode ini dibutuhkan ketika `KeyInfo` tidak dikirim sebagai bagian dari dokumen *Signed*XML :

Kode Program 13. Menge-*set* `KeyInfo`

```
SignedXml signedXml = new SignedXml();
DSA objDSAkeyPair = DSA.Create();
objDSAkeyPair.FromXmlString(strDSAKeyPairXML);
KeyInfo keyInfo = new KeyInfo();
keyInfo.AddClause(new DSAKeyValue(objDSAkeyPair));
signedXml.KeyInfo = keyInfo;
```

Kode ini digunakan untuk me-*load* dokumen *Signed*XML :

Kode Program 14. Me-*load* *Signed*XML

```
XmlDocument xmlDocument = new XmlDocument();
xmlDocument.PreserveWhitespace = true;
```

```
xmlDocument.LoadXml(signedxmldata);
```

Method `CheckSignature` pada `SignedXML` akan mengecek nilai *signature*. Kode ini digunakan untuk memverifikasi *Signature* :

Kode Program 15. *method* `CheckSignature`

```
XmlNodeList nodeList = xmlDocument.GetElementsByTagName("Signature");
signedXml.LoadXml((XmlElement)nodeList[0]);
if (signedXml.CheckSignature())
{
     MessageBox.Show("Signature check OK");
}
     else
{
     MessageBox.Show("Signature check FAILED");
}
```

Berikut adalah implementasi modul **Authentication** lengkapnya :

Kode Program 16. Authentication.cs

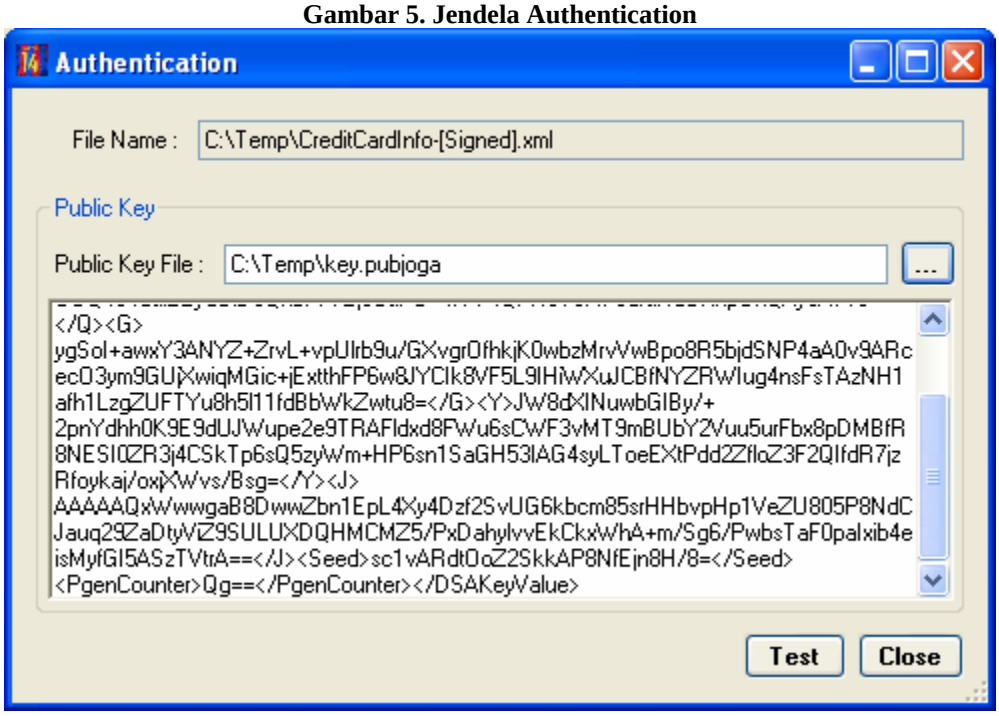
```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using System.IO;
using System.Security.Cryptography;
using System.Security.Cryptography.Xml;
using System.Xml;

namespace WindowsApplication1
{
    public partial class Authentication : Form
    {
        public Authentication()
        {
            InitializeComponent();
        }

        public Authentication(String path)
        {
            InitializeComponent();
            FileNameTextBox.Text = path;
        }

        public String getFileAsString(String path)
        {
            String result = "";
            try
            {
                using (System.IO.StreamReader sr = new
System.IO.StreamReader(path))
```

```
                {
                    result = sr.ReadToEnd(); //Membaca sampai EoF
                }
            }
            catch (Exception e)
            {
                MessageBox.Show("Error! \r \nFile not found");
                Console.Write(e.ToString());
            }
            return result;
        }

        private void OpenPublicKeyButton_Click(object sender, EventArgs
e)
        {
            if (openFileDialog1.ShowDialog() == DialogResult.OK)
            {
                String keypublic;
                PublicKeyTextBox.Text = openFileDialog1.FileName;
                keypublic = getFileAsString(PublicKeyTextBox.Text);
                PublicKeyTextBox.ReadOnly = true;
                KeyTextBox.ReadOnly = true;
                KeyTextBox.Text = keypublic;
                TestButton.Enabled = true;
            }
        }

        private void TestButton_Click(object sender, EventArgs e)
        {
            XmlDocument objDSAKeyPairXMLdocument = new XmlDocument();
            objDSAKeyPairXMLdocument.Load(PublicKeyTextBox.Text);
            string strDSAKeyPairXML =
objDSAKeyPairXMLdocument.InnerXml;
            String signedxmldata =
getFileAsString(FileNameTextBox.Text);

            // Create a SignedXml.
            SignedXml signedXml = new SignedXml();

            DSA objDSAkeyPair = DSA.Create();
            objDSAkeyPair.FromXmlString(strDSAKeyPairXML);
            KeyInfo keyInfo = new KeyInfo();
            keyInfo.AddClause(new DSAKeyValue(objDSAkeyPair));
            signedXml.KeyInfo = keyInfo;

            // Load the XML.
            XmlDocument xmlDocument = new XmlDocument();
            xmlDocument.PreserveWhitespace = true;
            xmlDocument.LoadXml(signedxmldata);

            XmlNodeList nodeList =
xmlDocument.GetElementsByTagName("Signature");
            signedXml.LoadXml((XmlElement)nodeList[0]);

            if (signedXml.CheckSignature())
            {
                MessageBox.Show("Signature check OK");
```

```
            }
            else
            {
                MessageBox.Show("Signature check FAILED");
            }
        }

        private void CloseButton_Click(object sender, EventArgs e)
        {
            this.Close();
        }
    }
}
```

Berikut adalah tampilan jendela **Authentication** dengan menggunakan file CreditCardInfo-[Signed].xml untuk diverifikasi dengan menggunakan kunci publik **key.pubjoga** :

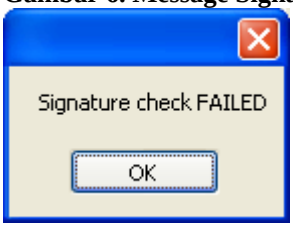
Gambar 5. Jendela Authentication



Terdapat dua kasus yaitu bila tidak terjadi modifikasi pada dokumen *Signed*XML (CreditCardInfo-[Signed].xml) dan bila terjadi modifikasi :

Gambar 6. Message Signature check OK

Gambar 6. Message Signature check Failed



Gambar 6 akan muncul bila terjadi modifikasi terhadap nilai elemen dalam CreditCardInfo-[Signed].xml. Modifikasi dapat berupa pengahapusan, penambahan, dan pengubahan nilai elemen dalam *Signed*XML.

9. Kesimpulan

Kesimpulan yang dapat diambil dari studi dan implementasi XML *Siganture* dalam Microsoft .NET ini adalah sebagai berikut :

a. XML *Signature* merupakan sebuah standar yang dikeluarkan oleh W3C, dapat digunakan sebagai salah satu alternatif dalam otentikasi dokumen digital, baik pada dokumen XML maupun jenis dokumen yang lain.
b. XML *Signature* mendukung penggunaan berbagai algoritma seperti RSA, DSA, PGP, dan juga standar-standar lain yang dikeluarkan oleh W3C.
c. Struktur XML *Signature* memiliki beberapa elemen yang wajib tercantum maupun yang opsional dalam sebuah dokumen XML. Tiap elemen mempunyai skema yang berbeda satu sama lain.
d. Terdapat beberapa algoritma yang digunakan dengan spesifikasi XML digital signature, antara lain Digest, base54, MAC, DSA, Canonicalization, dan Transform. Algoritma ini diitentifikasi oleh `URI` yang muncul sebagai atribut elemen. Parameter yang digunakan misalnya `SignatureMethod`, `KeyInfo`, dan `CanonicalizationMethod`.
e. XML *Signature* dapat diimplementasikan dalam Microsoft .NET *framework*, dalam makalah ini dibuat sebuah perangkat lunak bernama **XJS** (**XML Joga Signature**) dengan menggunakan Visual C#. XJS mempunyai tiga fungsi utama, yaitu pembangkitan kunci *keypair* & publik, penandatanganan dokumen XML, dan Otentikasi dokumen XML yang telah di-*sign* (*Signed*XML).
f. Terdapat tiga *namespace* utama yang dipakai dalam implementasi XML *Signature* ini, yaitu **`System.Xml`**, **`System.Security.Cryptography`**, **dan** **`System.Security.Cryptography.Xml`**. Dalam ketiga *namespace* diatas terdapat berbagai *class* dan *method* yang mendukung implementasi XML *Signature*.

DAFTAR PUSTAKA

[1] IBM, *Introducing XML Canonical Form : Making XML Suitable for regression testing, digital signatures, and more*. http://www-128.ibm.com/developerworks/xml/library/xc14n/Introducing XML canonical form.htm, diambil tanggal 11 Desember 2006 pukul 12:13.

[2] Simon, Ed et. al. *An Introduction to XML Digital Signatures.* http://www.xml.com/pub/a/2001/08/08/xmldsig.html, diambil tanggal 11 Desember 2006 pukul 12:15.

[3] W3C Recommendation, *XML – Signature Syntax and Processing.* http://www.w3.org/TR/xmldsig-core/, diambil tanggal 11 Desember 2006 pukul 12:05.

[4] W3C Recommendation, *Canonical XML Version 1.0*, http://www.w3.org/TR/2001/REC-xmlc14n-20010315/REC-xml-c14n-20010315.htm, diambil tanggal 11 Desember 2006 pukul 12:16.