Pemrograman XML Security

Budi Susanto
email: budsus@gmail.com

1. Pendahuluan

Masalah keamanan selalu menjadi hal penting untuk diperhatikan dalam dunia bisnis untuk memberikan kepastian tentang kesatuan dari isi dan transaksi yang terjadi serta menyakinkan bahwa informasi digunakan sebagaimana mestinya, terutama jika semuanya itu dilakukan secara digital, misal dengan memanfaatkan teknologi Internet.

Teknologi internet yang merupakan kumpulan jaringan-jaringan *heterogen* yang saling terhubung dan memiliki karakter komputasi yang tersebar (*distributed system*), dimana infrastruktur perangkat keras dan perangkat lunak (antara lain sistem operasi dan aplikasi) pada masing-masing jaringan lokal dapat sangat beragam. Dengan keragaman tersebut memberikan keleluasaan bagi setiap pemakai atau jaringan untuk bebas memilih jenis produk teknologi yang akan digunakan dan juga berhak mengatur dirinya sendiri. Walau demikian, hal tersebut juga akan membutuhkan terlalu banyak yang harus di administrasi, terlalu banyak jenis aplikasi, terlalu banyak variasi, dan harus dapat menanggapi perubahan teknologi yang cepat, ketika hendak merancang suatu infrastruktur tunggal untuk dapat memenuhi semua kebutuhan secara efektif. Misalkan kebutuhan untuk membangun infrastruktur tunggal seluruh perpustakaan nasional di Indonesia, dengan tetap mempertahankan sistem yang sudah digunakan dimasing-masing perpustakaan.

Untuk itu dibutuhkan suatu perluasan standard baru, disamping TCP/IP, yang dapat mengadopsi perubahan kebutuhan, menggabungkan teknologi baru dengan teknologi yang sudah berjalan, dan dapat diterapkan secara modular untuk bagian-bagian yang diperlukan saja. Standard ini juga harus dapat bekerja sama dengan baik, bukan secara replikasi (dimana masing-masing site, misalnya menyimpan duplikasi data yang sama), dan juga harus dapat cocok dengan teknologi baru untuk dapat membentuk: sistem tersebar terbuka (*open distributed system*), penggabungan aplikasi, dan *content management*.

Salah satu standard baru yang ditujukan untuk menjawab kebutuhan-kebutuhan tersebut di atas, adalah **eXtensible Markup Language (XML)**. XML telah diadopsi secara luas untuk beragam aplikasi dan beragam tipe informasinya. Selain itu juga telah digunakan untuk membentuk protokol dasar sistem tersebar untuk menggabungkan aplikasi-aplikasi yang tersebar di Internet, seperti protokol *Web Services*. Teknologi ini mulai diumumkan oleh W3C (*World Wide Web Consortium*) pada September 1998 sebagai sebuah teknologi *Markup Language*. Pengembangan XML ditujukan untuk mengatasi keterbatasan yang terdapat pada HTML (*Hypertext Markup Language*) yang merupakan

dasar bagi layanan-layanan berbasis *web* yang sudah ada. Sedangkan kelebihan XML dibandingkan dengan HTML adalah kemampuan XML untuk menyajikan struktur informasi secara lebih dinamis dan tidak terbatas pada ketentuan-ketentuan *markup* yang statis.

Sebagai sebuah *Markup Language,* maka sebuah dokumen XML tersusun dari kumpulan teks biasa. Sedangkan maksud yang terkandung pada data dijelaskan melalui sebuah penanda yang biasa disebut *Tag*. Dengan menggunakan *tag* tersebut, maka sebuah dokumen XML dapat dengan mudah dibaca dan dimengerti oleh pengguna bahkan oleh program komputer.

Seiring dengan semakin luasnya penggunaan XML pada berbagai layanan di Internet, dimana penyebaran informasinya sebagian besar menggunakan infrastruktur jaringan umum, maka mulai muncul permasalahan mengenai kebutuhan akan keamanan data bagi informasi yang terkandung didalam sebuah dokumen XML. Hal ini mengingat bahwa sebuah dokumen XML hanya tersusun dari sekumpulan teks yang sangat mudah untuk dipahami oleh pengguna atau program komputer.

Berdasarkan kebutuhan tersebut, maka W3C (*World Wide Web Consortium*) berusaha mengembangkan beberapa spesifikasi tambahan untuk XML. Spesifikasi tersebut ditujukan untuk memampukan para pengguna untuk menggunakan fasilitas pengamanan data pada dokumen XML yang hendak didistribusikan. Sistem keamanan data yang terdapat pada spesifikasi XML tersebut dikenal dengan istilah *XML Security*.

Dokumen ini akan merangkum teknologi kunci *XML Security* dan memberikan pengenalan bagaimana standard terbuka ini dapat digabungkan menjadi satu secara bersama-sama dengan dokumen XML. Diharapkan dengan penjelasan-penjelasan konsep dasar tersebut dapat memberikan pemahaman dan bekal penting untuk dapat mengikuti perkembangan dari spesifikasi standard yang berhubungan dengan *XML Security* saat ini dan selanjutnya.

2. Dasar-dasar XML

XML adalah salah satu bahasa Markup Language yang merupakan penyerderhanaan dari SGML (*Standard Generalized Markup Language*). XML dikembangkan oleh W3C dengan tujuan untuk melengkapi atau mengatasi keterbatasan pada teknologi HTML yang telah menjadi dasar layanan berbasis web saat ini. Pada penggunaannya, XML memiliki dua fungsi yaitu sebagai format dokumen dan format pertukaran data pada sebuah sistem yang terdistribusi.

Saat ini XML memegang peranan penting bagi sebagian transaksi informasi yang dilakukan melalui internet, karena XML telah menjadi sebuah format struktur pertukaran data yang dilakukan antar *web service* di internet. Pertumbuhan ini tidak lepas dari berbagai kelebihan yang dimiliki format XML, antara lain:

- XML memungkinkan sebuah transaksi berjalan secara *connectionless oriented*, artinya pelaku transaksi tidak perlu untuk membangun suatu koneksi khusus yang terhubung secara *real time;*

- Format XML berbasis teks dan berstruktur bebas, sehingga mampu memberikan fleksibilitas bagi sistem yang memanfaatkannya;
- XML tidak terikat pada suatu lingkungan sistem tertentu, sehingga sangat sesuai diterapkan pada jaringan yang dibangun dari sitem operasi dan aplikasi yang berbeda-beda.

Setiap dokumen XML memiliki struktur yang berfungsi untuk menjelaskan data yang tersimpan di dalamnya. Struktur tersebut dibentuk dari sekumpulan markup yang biasa disebut *Element*. Sebuah e*lement* dibentuk dari sepasang tag yang mengapit data yang akan dijelaskan. Adapun contoh bentuk dari sebuah dokumen XML, bernama buku.xml, dapat ditunjukkan pada kode 1.

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE budsus:KoleksiBuku SYSTEM "buku.dtd">
<budsus:KoleksiBuku xmlns:budsus='http://lecturer.ukdw.ac.id/budsus'>
   <budsus:buku id="1">
      <budsus:judul>XML Pocket Reference</budsus:judul>
      <budsus:harga>89500</budsus:harga>
   </budsus:buku>
   <budsus:buku id="2">
      <budsus:judul>Windows NT SNMP</budsus:judul>
      <budsus:harga>45</budsus:harga>
   </budsus:buku>
</budsus:KoleksiBuku>
```

Kode 1. Contoh dokumen XML yang valid

Baris pertama dari contoh buku.xml adalah elemen deklarasi XML yang mendeskripsikan dokumen tersebut sebagai sebuah dokumen XML. Contoh sebuah elemen pada dokumen buku.xml di atas adalah elemen `<budsus:judul></budsus:judul>`. Elemen `<budsus:judul>` pada buku dengan id=”1” memiliki nilai “XML Pocket Reference”. Di sini diperlihatkan suatu nama elemen yang menggunakan *namespace* “budsus”, dimana identitas “budsus” menunjuk pada URI 'http://lecturer.ukdw.ac.id/budsus'. *Namespace* merupakan tambahan untuk spesifikasi XML, yang sifatnya bukan merupakan suatu keharusan untuk digunakan dalam suatu dokumen XML. Namun dengan *namespace* dapat membantu untuk menghindari elemen yang bertabrakan dengan tag lain yang sama. Jika elemen `<budsus:judul/>` tanpa namespace, dapat cukup ditulis dengan `<judul></judul>`. Namun dalam penerapannya, sebagian pesar beberapa aplikasi berbasis XML menggunakan *namespace*-nya masing-masing.

Sebuah Element diperbolehkan untuk memiliki satu atau lebih nilai Atribut, yang berfungsi sebagai informasi tambahan dari elemen tersebut. Setiap atribut memiliki nama dan nilai dan diletakkan pada tag pembuka. Pada dokumen buku.xml, elemen `<budsus:buku>` memiliki sebuah atribut bernama ‘id’ dengan nilai ‘1’ dan '2'.

Dokumen buku.xml di atas merupakan contoh dokumen XML yang *valid*, dimana dalam dokumen tersebut terdapat deklarasi `<!DOCTYPE budsus:KoleksiBuku SYSTEM "buku.dtd">`, yang menyatakan bahwa struktur XML untuk `budsus:KoleksiBuku` harus mengikuti definisi DTD (*Document Type Definition*) yang tersimpan pada file buku.dtd. Selain DTD, terdapat bahasa lain yang dapat digunakan untuk mendefinisikan struktur dokumen XML, antara lain XDR (*XML-data Reduced Language*) dan XSD (*XML Schema Definition*). Definisi struktur dan contraints terhadap struktur dokumen XML dapat diletakkan secara internal ataupun eksternal. Seperti pada contoh buku.xml di atas, definisi DTD didefinisikan secara eksternal. Berikut contoh definisi struktur untuk elemen root `<budsus:KoleksiBuku>` yang tersimpan dalam file buku.dtd:

```
<!ELEMENT budsus:KoleksiBuku   (budsus:buku+)>
<!ELEMENT budsus:buku          (budsus:judul, budsus:harga)>
<!ATTLIST buku  id CDATA #REQUIRED>
<!ELEMENT budsus:judul         (#PCDATA)>
<!ELEMENT budsus:harga         (#PCDATA)>
```

Keuntungan dengan adanya definisi DTD, XDR atau XSD antara lain agar struktur yang ada pada dokumen XML dapat tetap terjaga pada saat dokumen XML didistribusikan. Selain itu salah satu syarat agar dokumen dianggap *valid* adalah bila dokumen tersebut sudah merupakan dokumen *well-formed*, yang merupakan bentuk dasar dari sebuah dokumen XML. Dokumen XML *well-formed* harus mengikuti syarat-syarat antara lain, sebagai berikut:

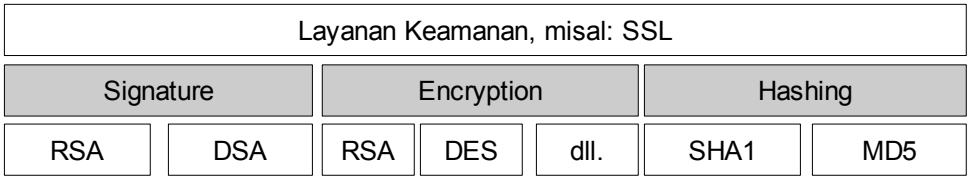
- setiap elemen sudah terdiri dari pasangan tag pembuka dan tag penutup;
- sudah benar dalam cara penulisan elemen, atribut, nilai;
- dalam sebuah dokumen harus terdapat sebuah root elemen yang menjadi puncak dari struktur yang dibentuk oleh elemen-elemen lainnya.

Sekali lagi dari pembahasan yang sangat singkat tentang dasar-dasar XML ini, dapat menegaskan kembali bahwa bahasa XML adalah berbasis teks dan dirancang untuk dapat diperluas dan dikombinasikan dengan standard lain. Dengan karakteristik ini, mestinya secara alamiah dapat menyediakan integritas, kerahasiaan dan keuntungan keamanan lain untuk seluruh atau sebagian dokumen XML dengan tetap menjaga agar dapat diproses oleh tool-tool standard XML. Oleh karena itu XML Security harus diintegrasikan dengan XML untuk tetap menjaga kemampuan XML selama penambahan kemampuan dalam keamanannya. Secara khusus hal tersebut penting untuk protokol berbasis XML, seperti SOAP (*Simple Object Access Protocol*), yang secara ekplisit dirancang untuk mengantarkan pesan.

3. Keamanan Data

Terhadap kebutuhan untuk dapat melakukan komunikasi ataupun transaksi secara aman pada jaringan Internet, teknik-teknik keamanan terus dikembangkan untuk semua lapisan pada arsitektur

protokol yang digunakan, khususnya arsitektur protokol TCP/IP. Pada lapisan di bawah aplikasi, Secure Socket Layers (SSL) telah menjadi alat keamanan yang sangat diminati oleh para pemakai, karena dapat membentuk saluran komunikasi yang aman antara dua titik akhir (*node*) yang saling berkomunikasi. SSL sebagai sebuah layanan keamanan, seperti layanan keamanan lainnya, secara umum dapat dipastikan akan dibangun dengan menerapkan beberapa mekanisme keamanan, seperti tandatangan (*signature*), penyandian (*Encryption*), dan *hashing*. Dari masing-masing mekanisme keamanan tersebut, akan diterapkan suatu algoritma. Gambar 1 memperlihatkan contoh model arsitektur layanan keamanan data.



Gambar 1. Layanan, Mekanisme dan Algoritma Keamanan Data

Walaupun demikian, SSL tidak melindungi apa yang terjadi sebelum dan sesudah titik akhir (*node*) komunikasi. Sehingga dalam hal ini sangat dibutuhkan penerapan teknik keamanan pada lapisan aplikasi. Metode yang terkenal untuk dapat menerapkan keamanan pada lapisan aplikasi antara lain *Pretty Good Privacy* (PGP) dan *Secure MIME* (S/MIME). Namun pada kenyataannya, kedua metode tersebut memerlukan aplikasi yang mendukung metode-metode tersebut. Selain itu juga kedua metode tersebut hanya dapat menyandikan (enkrip) sebuah dokumen keseluruhan atau tidak sama sekali. Kondisi ini tidak selalu dibutuhkan dan tidak selalu diharapkan. Sebuah dokumen mungkin berisi bagian-bagian yang harus dapat dibaca oleh siapa saja, dan ada bagian lain yang harus tetap dijaga kerahasiaannya. Arsitektur keamanan untuk XML (XML Security) dikembangkan untuk menjawab kebutuhan tersebut. Arsitektur keamanan XML Security dapat menyediakan keamanan yang fleksibel tidak hanya untuk dokumen XML, namun juga untuk semua objek yang dapat ditangani melalui sebuah URI.

Sebuah arsitektur keamanan dikembangkan dengan mendasari dirinya pada beberapa layanan keamanan berikut ini :

- Kerahasiaan (*Confidentiality*): sebuah dokumen rahasia seharusnya tidak dapat dibaca oleh orang yang tidak berwewenang. Hal ini dapat dipastikan dengan melakukan enkripsi/penyandian data terhadap dokumen tersebut.
- Integritas (*Integrity*): Integritas berarti modifikasi data yang tidak diijinkan dapat dideteksi. Integritas dapat diyakinkan dengan penghitungan *message digests*.

- *Authentication*: Otentikasi memastikan keaslian data atau identitas pasangan komunikasi. Otentikasi keaslian data dapat dicapai dengan perhitungan *digital signature* dan *message authentication code* (MAC).
- *Nonrepudiation*: atau anti penyangkalan mengkombinasikan integritas data dengan otentikasi keaslian data, sehingga pengirim data tidak dapat membantah modifikasi yang dia lakukan terhadap data.

Beberapa teknologi untuk menerapkan layanan-layanan tersebut di atas antara lain :

- ***Symmetric encryption keys***: metode enkripsi ini menggunakan kunci yang sama baik untuk enkripsi maupun dekripsi. Metode enkripsi ini cepat, namun oleh karena pengirim dan penerima memiliki kunci yang sama, pengiriman kunci menimbulkan resiko keamanan. Kunci dengan panjang minimal 128 bit dianggap sudah aman. Sebagai contoh AES (128–256 bit), Blowfish (64–448 bit), dan Twofish (128–256 bits).
- ***Asymmetric keys***: metode enkripsi ini menggunakan sepasang kunci: *public key* digunakan oleh pengirim untuk menyandikan pesan, dan *private key* (harus dijaga kerahasiaannya) digunakan oleh penerima untuk mendekrip pesan. Contoh kunci ini antara lain RSA (*Rivest-Shamir-Adleman*) dan ECC. Keuntungan dari kunci *asymmetric* adalah tidak memerlukan pertukaran kunci yang tidak aman.
- ***Hybrid keys***: teknik ini menggunakan kunci *symmetric* dan *asymmetric*. Langkah pertama, pengirim dan penerima saling menukarkan kunci symmetric namun menyandinya dengan kunci asymmetric. Komunikasi berikutnya dapat disandikan dengan kunci symmetric.
- ***Message digests***: algoritma m*essage digest* digunakan untuk menghitung nilai hash dari data yang harus dilindungi dari pengrusakan. Algoritma ini dapat berjalan dengan cepat dan menjamin bahwa setiap perubahan dalam pesan akan menyebabkan perubahan yang berarti terhadap nilai hash-nya. Untuk amannya, panjang nilai hash minimal adalah 160 bit. Algoritma yang terkenal antara lain MD5 (sudah akan dikategorikan sebagai tidak aman), SHA-1 (*Secure Hash Algorithm, Revision 1*), dan RIPEMD160 (*RACE Integrity Primitives Evaluation Message Digest*).
- ***Digital signatures***: menggunakan teknik *asymmetric* untuk menghasilkan sebuah tandatangan (*signature*) dari isi dokumen yang ditandatangani. Penandatangan menggunakan *private key* untuk menghasilkan nilai tandatangan, sedangkan pembaca menggunakan *public key* untuk menguji validitas dari tandatangan. Oleh karena algoritma *assymmetric* lambat, kemudian digunakan teknik *hybrid*. Pertama, sebuah *message digest* akan dihitung dari isi dokumen, kemudian nilai *digest* dilindungi dengan kunci *assymmetric*. Metode yang populer antara lain DSA (*Digital Signature Algorithm*) dan RSA.

- **Sertifikat**: masalah dengan *public key* adalah dimungkinkan pihak ketiga menangkap publikasi dari sebuah *public key* dan menggantinya dengan public key miliknya sendiri. Sehingga setelah itu, pihak ketiga akan dapat membaca setiap pesan yang tersandi dengan *public key* tersebut. Masalah ini dapat dipecahkan dengan menerbitkan sebuah sertifikat terhadap *public key*. Sebuah *certification authority* (CA) menandai *public key* dengan menanamnya pada sebuah sertifikat. Seorang pengirim yang ingin menggunakan *public key* dari penerima pesan dapat bertanya untuk sebuah sertifikat yang berisi *public key* tersebut. Untuk memeriksa keabsahan sertifikat, pengirim dapat memvalidasinya kembali dengan *public key* milik CA. Public keys milik CA di tuliskan dalam tubuh software client (*hard-coded*) seperti browser web atau program *signature*.

XML Security sebagai sebuah standar arsitektur layanan dan teknologi keamanan, sangat memerhatikan layanan-layanan yang dibutuhkan untuk "merasa" aman tersebut. Mengacu pada karakteristik XML yang ada, standar XML Security dikembangkan untuk memberikan keuntungan tersendiri dalam hal fleksibilitas, ekstensibilitas dan kemudahan dalam penerapannya.

4. Mengenal XML Security

Dokumen XML adalah sebuah dokumen teks (alphabetis, alphanumeris dan beberapa karakter simbol teks) yang tidak dapat menerima atau mempresentasikan format biner. Sehingga dalam pengembangan arsitektur XML Security tentunya tidak dapat langsung menggunakan teknologi dan algoritma keamanan lama yang menggunakan format biner sehingga memerlukan perangkat lunak khusus untuk menggunakan dan menterjemahkannya. Alasan lainnya karena standard algoritma dan teknologi keamanan lama tidak dirancang untuk digunakan dalam XML dan tidak mendukung pendekatan teknis XML untuk mengatur *content* di dalamnya, seperti *content* yang dinyatakan dengan *uniform resource identifier* (URI) atau menggunakan deifnisi standard XML lain untuk menyatakan lokasi bagian dari *content* XML (seperti XPath).

XML Security menyelesakan masalah-masalah tersebut dengan mendefinisikan sebuah kerangkat kerja dan aturan pemrosesan yang bersifat umum yang dapat dipakai bersama-sama oleh berbagai aplikasi dengan menggunakan tool yang banyak digunakan agar lebih memudahkan dalam penerapannya. XML Security juga dapat menggunakan ulang konsep, algoritma dan teknologi inti dari sistem keamanan yang sudah ada ketika memulai perubahan untuk penerapan XML di dalamnya, untuk menyediakan solusi yang praktis, fleksibel, dan ekstensibilitas. Dengan menggunakan teknologi, paradigma dan tool XML yang sudah ada, XML Security meminimalkan modifikasi aplikasi untuk dapat mencapai kebutuhan keamanan yang diinginkan.

Standard XML Security menyediakan sekumpulan standard teknis untuk memenuhi kebutuhan keamanan. Standard tersebut dirancang untuk menyesuaikan dengan paradigma XML. Dengan

dikembangkannya standard XML Security secara otomatis akan mempengaruhi dan sekaligus meningkatkan standard umum XML. Berikut di antaranya [http://www.sitepoint.com/authorcontact/245/933] :

- standard XML Security mendefinisikan kosa kata XML untuk mengambarkan informasi keamanan dengan menggunakan teknologi XML, seperti XML Schema. Sebagai contoh elemen `<KeyInfo>` didefinisikan dalam XML Digital Signature untuk menyatakan informasi kunci tersandi atau tertanda tangan. Definisi ini digunakan dalam sejumlah spesifikasi yang menggunakan kosa kata XML yang sama untuk arti yang serupa.
- Standard XML Security menggunakan standard XML lain yang sudah ada yang memungkinkan untuk meningkatkan atau mempengaruhi usaha-usaha XML saat ini. Sebagai contoh, XML Digital Signature dapat menerima ekspresi XPath untuk mengambil bagian tertentu dari XML untuk diproses.
- Standard XML Security dirancang untuk menawarkan aspek fleksibilitas dan ekstensibilitas dari XML. Standard XML Security dapat diterapkan pada suatu dokumen XML keseluruhan, atau hanya pada elemen ataupun pada nilai elemen XML. Standard XML Security mendukung perluasan kosa kata (elemen) XML dengan cara menggunakan XML namespace dan definisi perluasan XML Schema.
- XML Security dapat diterapkan untuk keamanan lapisan aplikasi antar dua node, terutama ketika pesan XML dirutekan melalui beberapa pemroses perantara. Disamping itu, XML Security juga dapat digunakan dengan teknologi keamanan transport, seperti SSL/TLS.
- XML Security sedapat mungkin menerapkan ulang kriptografi dan teknologi yang sudah ada tanpa membuat atau menciptakan ulang. Sebagai contoh standard format sertifikat X.509 V3 digunakan tanpa pendefinisian ulang ketika digunakan; algoritma digest seperti SHA1 juga digunakan dalam standard XML Security dengan menyertakan identitas URI yang unik untuk algoritma tersebut dan mendefinisikan bagaimana algoritma tersebut digunakan dalam model pemrosesan XML Security.

Standard XML Security ini memiliki beberapa standard inti yang dirancang untuk menyediakan teknologi yang kompatibel dengan XML untuk dapat memenuhi kebutuhan keamanan. Standard-standard inti tersebut kemudian sebagian besar diterapkan untuk beberapa spesifikasi lain seperti *Web Services*, *Digital Rights Management - eXtensible Rights Markup Language 2.0 (XrML)*, *Privacy - Platform for Privacy Preferences (P3P)* dan *ebXML*.

Berikut standar-standar inti dari XML Securiti [4, 5] :

- XML Digital Signature untuk tanda tangan dan integritas,
- XML Encryption untuk kerahasiaan,

- kemudian berdasar kedua standard tersebut terdapat standard lain sebagai komponen inti dari arsitektur XML Security, yaitu:
    - XML Key Management (XKMS) untuk manajemen kunci,
    - Security Assertion Markup Language (SAML) untuk pembuatan pernyataan otentikasi dan otorisasi, dan
    - XML Access Control Markup Language (XACML) untuk memulai aturan otorisasi.

Standard XML Digital Signature dan XML Encryption merupakan dua standard pokok dalam arsitektur XML Security, karena keduanya akan dipakai juga dalam komponen arsitektur XML Security. Terutama sangat direkomendasikan untuk XML Digital Signature digunakan, karena standard tersebut memberikan elemen-elemen yang dapat dipakai bersama pada standard lainnya (misal `<KeyInfo>`). Dengan dapat dipakai bersama-sama `<KeyInfo>`, maka pada standar-standar yang lain dapat menggunakan digital signature untuk memastikan integritas dari dokumen XML.

Setiap standar tersebut di atas mendefinisikan kosa kata elemen-elemen XML untuk menjelaskan informasi keamanan yang dibutuhkan untuk aspek keamanan yang terkait, termasuk aturan-aturan pemrosesan yang diperlukan untuk dapat memahami tentang bagaimana menerapkan standar tersebut.

Dalam tulisan ini hanya dipusatkan pada spesifikasi XML *Digital Signature* dan XML *Encryption*, mengingat bahwa kedua spesifikasi tersebut merupakan inti terdalam dalam standard arsitektur XML Security.

5. XML Digital Signature

Sebuah *digital signature* yang ada, seperti RSA dan DSA, akan menghasilkan sederetan keluaran biner terhadap sembarang *content* digital. Keluaran dari tandatangan RSA terkait dengan ukuran kunci yang digunakan, sedangkan keluaran DSA terkait dengan representasi pengkodean yang digunakan. Disamping itu, untuk memeriksa tandatangan digital, penandatangan harus menyediakan informasi tambahan kepada pemeriksa, termasuk di dalamnya tipe algoritma yang digunakan, informasi tentang penerima dan juga kunci pemeriksaan/pembuktian.

XML Signature (Tandatangan digital XML) adalah sebuah spesifikasi sintak XML yang digunakan untuk mempresentasikan suatu *digital signature* terhadap sembarang *content* digital, misalnya data karakter tersandi (misal HTML), data biner tersandi (misal JPEG), data XML tersandi, atau sebuah bagian dalam suatu dokumen XML. XML Signature merupakan tandatangan digital yang dirancang untuk digunakan pada transaksi XML.

Karakter yang menonjol dari XML Signature adalah kemampuannya dapat menandatangani hanya bagian tertentu dari XML daripada keseluruhan dokumen. Fleksibilitas ini dapat dimanfaatkan

untuk memastikan kesatuan bagian tertentu dari dokumen XML, selagi tetap membuka bagian lain dokumen untuk dapat diubah.

Untuk dapat melakukan validasi tandatangan, objek data yang ditandatangani harus dapat diakses. XML Signature sendiri akan membangkitkan lokasi dari objek data asli yang tertandatangani dengan salah satu kemungkinan berikut ini :

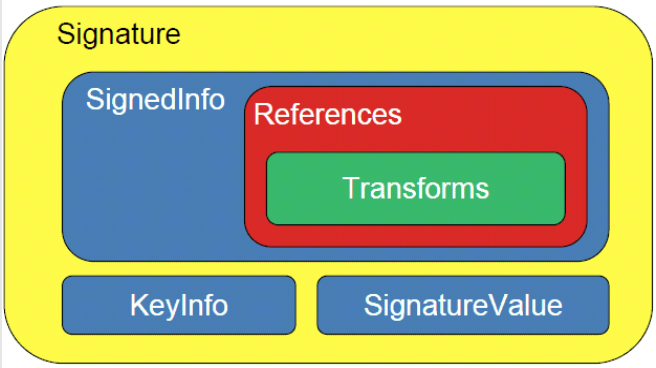
- dapat direferensikan dengan suatu URI di dalam XML signature;
- terletak dalam elemen sesumber yang sama sebagai XML signature;
- ditanamkan dalam XML signature (*signature* sebagai elemen induk);
- memiliki XML signature yang ditanamkan dalam dirinya sendiri (*signature* sebagai elemen anak).

Dengan beberapa uraian singkat tersebut, kita akan mulai mengenal lebih detail terhadap struktur XML Signature termasuk elemen-elemen pentingnya yang perlu diketahui.

5.1 Struktur XML Signature

Seperti yang ditulis pada kode 2., elemen-elemen yang digunakan adalah tag-tag XML, dan strukturnya mendefinisikan hubungan induk-anak. Pada kode 2. tersebut juga memunculkan beberapa simbol operator *cardinality* yang menyatakan jumlah kemunculan dari tiap elemen di dalam elemen induk `<Signature>`. Definisi untuk tiap operator cardinality tersebut, antara lain: "?" menyatakan satu atau tidak ada; "+" menyatakan satu atau lebih kemunculan; dan "*" menyatakan tidak ada atau jika ada dapat lebih dari satu kali kemunculan. Ketiadaan suatu operator cardinality pada sebuah elemen atau atribut menyatakan bahwa kemunculan elemen tersebut hanya satu kali.

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



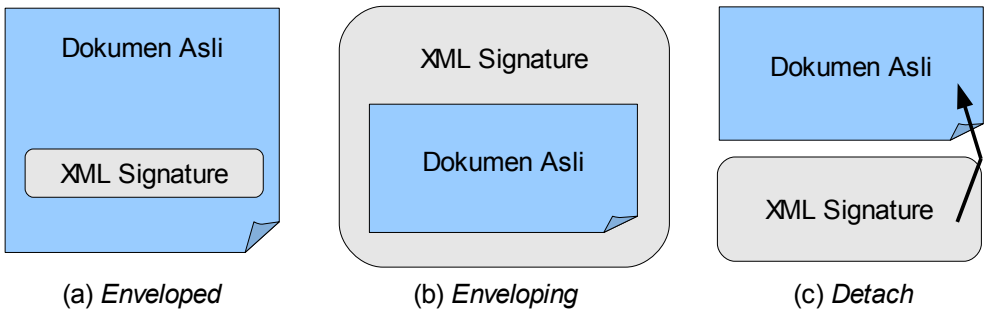
Kode 2. Struktur dasar XML Signature

Sebuah XML Signature dimulai dengan elemen induk <Signature> yang menyediakan struktur dan pengenal (ID) untuk tandatangan. Elemen induk <Signature> berisi dua elemen anak, yaitu <SignedInfo> dan <SignatureValue>. Elemen <SignedInfo> berisi daftar dari sesuatu yang ditandatangani, termasuk di dalamnya informasi tentang tandatangan, seperti algoritma tandatangan digital yang digunakan, yang didefinisikan dalam elemen <SignatureMethod>. Sesumber data yang ditandatangani dinyatakan dengan elemen <Reference>, dan sintak URI digunakan untuk menunjuk pada lokasi aliran data yang dilakukan *digest* dan ditandatangani. Elemen <CanonicalizationMethod> menyatakan algoritma yang digunakan untuk menyederhanakan bentuk (*canonicalize*) elemen <SignedInfo> sebelum disarikan sebagai bagian dari operasi tandatangan. Kedua elemen <CanonicalizationMethod> dan <SignatureMethod> digunakan dalam proses pembangkitan <SignatureValue>.

Elemen <KeyInfo> dapat digunakan untuk membantu memfasilitasi pemrosesan otomatis XML Signatures dengan menyediakan suatu mekanisme untuk pengenalan kunci verifikasi. Elemen <Object> merupakan elemen yang berisi sembarang tipe objek data. Dua tipe khusus yang direkomendasikan oleh XML Signature dalam <Object> adalah elemen <SignatureProperties> dan <Manifest>. Elemen <SignatureProperties> berisi pernyataan tegas tentang tandatangan yang digunakan. Pernyataan tersebut bermanfaat untuk meningkatkan kepercayaan dalam mekanisme validasi tandatangan dan integritas data. Elemen <Manifest> digunakan untuk memecahkan dua masalah: menyediakan referensi validasi untuk aplikasi dan menyediakan peran yang sesuai bagi beberapa penandatangan untuk menandatangani beberapa dokumen. Tanpa elemen <Manifest>, hasil dari tandatangan akan lebih besar, memiliki semantik yang berulang-ulang (*redundan*) dan akan membebani selama proses pembuatan dan verifikasi.

5.2 Tipe XML Signature

Sebelum kita membahas lebih jauh tentang beberapa elemen penting dalam XML Signature, bagian ini akan didahului dengan beberapa tipe dasar tandatangan terkait dengan hubungan induk dan anak. Antara satu aplikasi dengan aplikasi lain mungkin memerlukan penyampaian tandatangan dengan cara tertentu, atau disebut sebagai *tipe tandatangan*. Aplikasi tertentu memerlukan bahwa tandatangan merupakan bagian dari dokumen asli yang terkirim, dan ada pula yang memproses data asli terpisah dari tandatangan dan mungkin antara data dan tandatangan harus terpisah. Dokumen asli yang tergabung dengan elemen <Signature> (dokumen asli sebagai induk atau anak dari elemen <Signature>), disebut sebuah tandatangan tersampul (*enveloped*) atau penyampul (*enveloping*). Sedanghkan dokumen asli yang terpisah dari elemen <Signature> (dokumen asli tidak memiliki hubungan induk-anak dengan elemen <Signature>) disebut sebagai tandatangan terpisah (*detach*). Ilustrasi dari ketiga tipe XML Signature dapat ditunjukkan pada gambar 2.



Gambar 2. Tipe XML Signature

5.3 Sintak XML Signature

Berikut akan diuraikan tentang beberapa elemen penting yang digunakan dalam XML Signature, mengingat elemen-elemen tersebut perlu mendapat perhatian khusus dalam pemakaiannya. Walaupun singkat, diharapkan uraian yang diberikan cukup memberikan pemahaman.

Elemen `<SignatureMethod>`

`<SignatureMethod>` merupakan elemen yang menyatakan algoritma yang digunakan untuk pembangkitan dan validasi tandatangan. Algoritma-algoritma tersebut mengidentitaskan semua fungsi kriptografi yang terpakai dalam operasi tandatangan (antara lain hashing, algoritma public key, MAC, padding, dan sebagainya). Algoritma-algoritma tersebut dinyatakan dengan suatu identitas URI antara lain :

- HMAC-SHA1 (http://www.w3.org/2000/09/xmldsig#hmac-sha1),
- DSAwithSHA1 (http://www.w3.org/2000/09/xmldsig#dsa-sha1),
- RSAwithSHA1 (http://www.w3.org/2000/09/xmldsig#rsa-sha1).

Contoh penulisan elemen ini :

```
<SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#hmacsha1">
  <HMACOutputLength>80</HMACOutputLength>
</SignatureMethod>
```

atau

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

Elemen `<CanonicalizationMethod>`

Canonicalization digunakan untuk memastikan bahwa XML ditangani secara konsisten oleh berbagai pemroses XML secara khusus terhadap white space (spasi, tabulasi) dan variasi lain. Dengan algoritma *canonical* yang dikembangkan oleh IETF dan W3C[1], dapat menghindari masalah

[1]Standarisasi Canonical XML dapat dilihat di http://www.w3.org/TR/2001/REC-xml-c14n-20010315

ketidak konsistennya penulisan dokumen XML. Sebagai contoh antara <Signature Id="TtdPertamaku"> dengan <Signature Id="TtdPertamaku"> akan menghasilkan deretan string oktet yang berbeda jika di lakukan message digest dengan algoritma SHA-1, padahal keduanya secara semantic menunjuk pada elemen dan arti yang sama. Canonicalization secara sederhana memastikan bahwa oktet yang ditandatangani sama dengan nilai hash untuk memastikan validitas tandatangan. Contoh pemakaian elemen ini :

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

Elemen <Reference>

Elemen ini digunakan untuk menjelaskan bagaimana sesumber diperoleh dan ditransformasikan untuk menghasilkan data yang disarikan dan ditandatangani sebagai bagian dari elemen <SignedInfo>. Elemen <Reference> berisi tiga elemen anak <Transforms>, <DigestValue>, dan <DigestMethod>, dan juga tiga atribut pilihan: Id, Type, dan URI. Atribut Id menyatakan identitas unik untuk suatu referensi. Atribut Type menunjuk tipe referensi yang ditunjuk. Ada dua nilai identitas URI yang direkomendasikan, yaitu http://www.w3.org/2000/09/xmldsig#Object, http://www.w3.org/2000/09/xmldsig#SignatureProperties dan http://www.w3.org/2000/09/xmldsig#Manifest.

Dalam kasus sederhana sebuah elemen <Reference> hanya memerlukan elemen <DigestMethod> dan <DigestValue>. Untuk algoritma *digest* yang direkomendasikan pada XML Signature hanya satu, yaitu SHA-1. Berikut contoh elemen <DigestMethod>:

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

Berikut ini merupakan sebuah contoh bagaimana pemakaian elemen <Reference> yang tidak tidak menyebutkan atribut Type, karena data yang ditandatangani dinyatakan pada elemen <Object> dalam dokumen XML Signature yang sama.

```
<Signature>
  <SignedInfo>
    <Reference>
      <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">
    ...(deretan string terenkodekan base64)...
  </Object>
  ...
</Signature>
```

Elemen <Reference> di atas, sama jika ditulis dengan menyertakan atribut Type berikut :

```
<Reference Type="http://www.w3.org/2000/09/xmldsig#Object" URI="#ImportantPicture">
  ...
```

```
</Reference>
```

Sebuah contoh lain (lihat kode 3) [4, p. 139] menunjukkan pemakaian tipe `SignatureProperties` dan sekaligus menunjukkan bagaimana XML Signature dapat menyatakan multi tandatangan untuk multi data.

```
<Signature Id="SignedCheckToPaperBoy">
  <SignedInfo>
    <Reference URI="#CheckToPaperBoy">
      <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
      <DigestValue>3846JEYbJymGoDfgMRaH5PYeNQv=</DigestValue>
    </Reference>
    <Reference URI="#FictionalSignatureAssertions"
       Type="http://www.w3.org/2000/09/xmldsig#SignatureProperties">
      <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
      <DigestValue>r3653rvQTO0gKtMyu4VfeVu9ns=</DigestValue>
    </Reference>
  </SignedInfo>
  <Object>
    <ElectronicCheck Id="CheckToPaperBoy">
      <RecipientName>PaperBoy</RecipientName>
      <SenderName>L.Meyer </SendName>
      <AccountNumber>765121-2420</AccountNumber>
      <Amount>$2</Amount>
    </ElectronicCheck>
  </Object>
  <Object>
    <SignatureProperties>
      <SignatureProperty Id="FictionalSignatureAssertions"
                         Target="#SignedCheckToPaperBoy">
        <Assertion>
          <GenerationTime>Mon Jun 11 19:10:27 UTC 2001</GenerationTime>
        </Assertion>
        <Assertion>
          <Note> Can only be cashed at Bank Foobar </Note>
        </Assertion>
        <Assertion>
          <ValidityDays> 90 </ValidityDays>
        </Assertion>
      </SignatureProperty>
    </SignatureProperties>
  </Object>
</Signature>
```

Kode 3. Contoh XML Signature Multi Reference dan SignatureProperties

Contoh terakhir (lihat Kode 4 [4, p. 142]) adalah referensi yang bertipe Manifest. Manifest sendiri merupakan sebuah elemen anak dari elemen `<Object>` yang menyatakan kumpulan beberapa referensi data yang ditandatangani.

```
<Signature Id="ManifestExample">
  <SignedInfo>
    <Reference URI="#ReportList" Type="http://www.w3.org/2000/09/xmldsig#Manifest">
      <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
      <DigestValue>545x3rVEyOWvKfMup9NbeTujUk=</DigestValue>
    </Reference>
  </SignedInfo>
  <Object>
    <Manifest Id="ReportList">
      <Reference URI="http://www.myserver.com/Report.pdf">
        <DigestMethod
              Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
        <DigestValue>20BvZvrVN498RfdUsAfgjk7h4bs=</DigestValue>
      </Reference>
      <Reference URI="http://www.myserver.com/Report.gif">
        <DigestMethod
              Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
        <DigestValue>40NvZfDGFG7jnlLp/HF4p7h7gh=</DigestValue>
      </Reference>
    </Manifest>
  </Object>
  ...
</Signature>
```

Kode 4. Contoh tipe referensi data Manifest

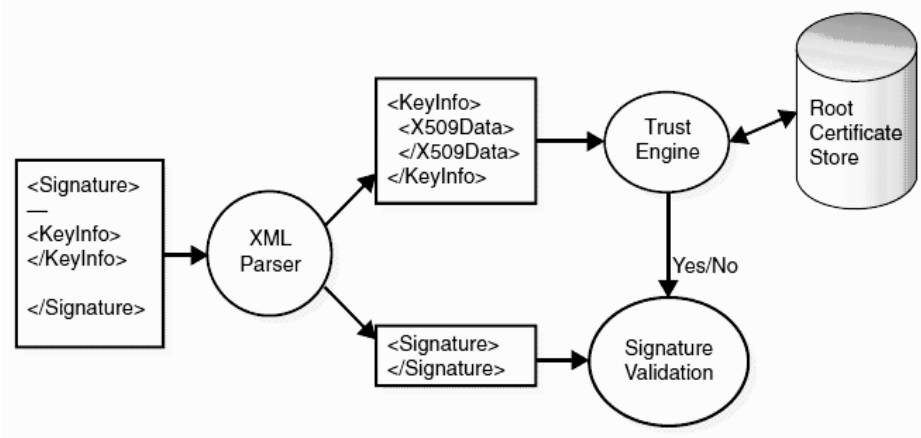
Elemen `<KeyInfo>`

Elemen ini bersifat pilihan, yang dapat memberikan peran pengujian integritas XML Signature dalam suatu aplikasi. Dengan informasi yang tertulis dalam elemen `<KeyInfo>` ini, memungkinkan penerima untuk menguji tandatangan tanpa perlu memiliki atau mencari kunci verifikasi. Elemen ini dapat membantu verifikasi tandatangan secara otomatis, namun juga sekaligus dapat berbahaya. Elemen ini sudah diluar dari sintak tandatangan dan masuk dalam daerah aplikasi.

Aplikasi penerima XML Signature harus mengetahui kapan mempercayai informasi dalam `<KeyInfo>` dan kapan tidak. Salah satu caranya adalah dengan menggunakan mesin penguji kepercayaan yang memproses informasi dalam elemen `<KeyInfo>`, dan membuat keputusan kepercayaan berdasarkan isinya. Gambar 3 memberikan ilustrasi bagaimana sebuah dokumen XML yang berisi elemen `<Signature>` dapat diterjemahkan untuk menerima elemen `<KeyInfo>`. Pada contoh tersebut elemen `<KeyInfo>` berisi sertifikat berformat X.509 yang kemudian dilewatkan ke mesin penguji kepercayaan. Sertifikat dalam `<KeyInfo>` diuji dengan sertifikat root yang tersimpan pada penyimpan terpercaya. Konsep mesin penguji ini merupakan salah satu definisi spesifikasi dari XKMS.

Selain mekanisme validasi melalui jalur sertifikat tersebut, pada elemen `<KeyInfo>` menyediakan beberapa elemen anak yang dapat digunakan untuk mekanisme validasi XML Signature

(lihat Tabel 1). Tidak semua elemen yang ada tersebut diperlukan untuk penerapan XML Signature. Hanya elemen `<KeyValue>` yang diperlukan, dan `<RetrievalMethod>` direkomendasikan.



Gambar 3. Contoh layanan pengujian kepercayaan

Tabel 1. Elemen anak <KeyInfo>

Nama Elemen	***Keterangan***
<KeyName>	Identitas nama kunci
<KeyValue>	Public key RSA atau DSA
<RetrievalMethod>	Referensi informasi kunci. Beberapa nilai atribut `Type` yang dapat digunakan: * 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 * http://www.w3.org/2000/09/xmldsig#rawX509Certificate
<X509Data>	Sertifikat X.509, nama, dan data terkait lainnya. Elemen-elemen anak:

Elemen	***Keterangan***

<X509IssuerSerial>	Penerbit dan nomor serialnya
	Elemen anak:
	`<X509IssuerName>`
	`<X509SerialNumber>`

<X509SubjectName>	X.509 nama subjek
<X509Certificate>	Sertifikat X.509v3
<X509SKI>	X.509 SubjectKeyIdentifier
<X509CRL>	X.509 Certificate Revocation List

<PGPData>	Kunci PGP dan identitasnya

Nama Elemen	*Keterangan*
<SPKIData>	Kunci SPKI, sertifikat, atau data SPKI lain
<MgmtData>	Parameter persetujuan kunci (seperti parameter Diffie-Hellman)

Berikut contoh elemen <KeyInfo> yang berisi sertifikat X.509 dengan beberapa elemen anaknya sebagai uraian rinci sertifikat X.509 yang digunakan untuk validasi (Kode 5).

```
<Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
   <SignedInfo>
     ...
   </SignedInfo>
   <SignatureValue>
     ...
   </SignatureValue>
   <KeyInfo xmlns="http://www.w3.org/2000/09/xmldsig#">
     <X509Data>
       <X509IssuerSerial>
         <X509IssuerName>CN=John Sheridan,OU=Babylon 5,O=Interstellar
Alliance,L=Babylon,ST=Babylon,C=B5</X509IssuerName>
         <X509SerialNumber>1031403982</X509SerialNumber>
       </X509IssuerSerial>
       <X509SubjectName>CN=John Sheridan,OU=Babylon 5,O=Interstellar
Alliance,L=Babylon,ST=Babylon,C=B5</X509SubjectName>
       <X509Certificate>
         ... (deretan string base64)...
       </X509Certificate>
     </X509Data>
   </KeyInfo>
</Signature>
```

Kode 5. Contoh pemakaian elemen <KeyInfo>

5.4 Pembangkit Inti (*Core Generation*)

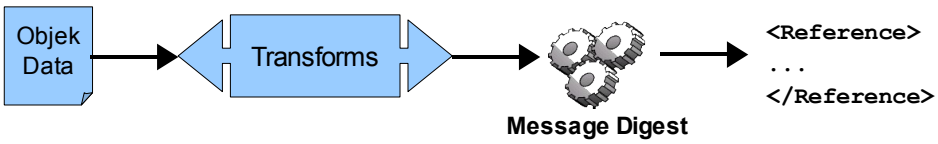
Spesifikasi XML Signature mendefinisikan core generation dalam dua langkah: *reference generation* (pembangkit referensi) dan *signature generation* (pembangkit tandatangan). Langkah pertama mendefinisikan bagaimana nilai *digest* untuk setiap `<Reference>` dihitung serta membuat struktur lengkapnya; dan langkah kedua mendefinisikan bagaimana nilai `<SignatureValue>` dihitung serta sekaligus struktur blok elemen induk `<Signature>`.

Berikut beberapa tahapan proses *reference generation* (gambar 4):

1. terapkan `<Transforms>` ke objek data.
2. Hitung nilai *digest* terhadap objek data yang dihasilkan.
3. Buat elemen `<Reference>`, termasuk identitas objek data (pilihan), sembarang elemen transform (pilihan), algoritma digest dan DigestValue.

Setelah itu, berikut tahapan proses *signature generation* (gambar 5):

1. buat elemen `<SignedInfo>` dengan `<SignatureMethod>`, `<CanonicalizationMethod>` and `<Reference>`.
2. *Canonicalize* dan hitung `<SignatureValue>` berdasar algoritma yang tersebut pada `<SignedInfo>`.
3. Bentuk elemen `<Signature>`, termasuk `<SignedInfo>`, `<Object>`, `<KeyInfo>` (jika diperlukan), dan `<SignatureValue>`.

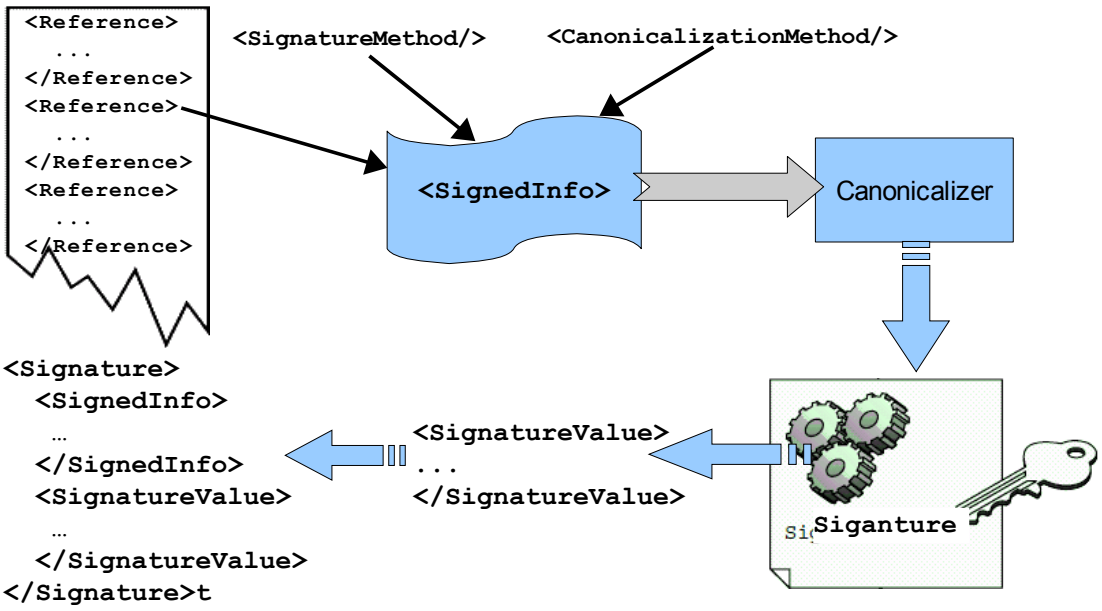


Gambar 4. Reference Generation

Transformasi yang dilakukan terhadap setiap objek data dilakukan untuk mendapatkan bentuk data yang canonical agar siap untuk diproses dengan algoritma *digest*. Untuk transformasi ini, XML Signature merekomendasikan 5 metode transformasi, antara lain :

1. Optional XSLT (http://www.w3.org/TR/1999/REC-xslt-19991116)
2. Recommended Xpath (http://www.w3.org/TR/1999/REC-xpath-19991116)
3. Required Enveloped Signature* (http://www.w3.org/2000/09/xmldsig#enveloped-signature)
4. Base64 Decode (http://www.w3.org/2000/09/xmldsig#base64)
5. Canonical XML (http://www.w3.org/TR/2001/REC-xml-c14n-20010315)

Keluaran dari setiap `<Transform>` diperlakukan sebagai masukan untuk <Transform> berikutnya. Masukan dari `<Transform>` pertama merupakan hasil dari dereferencing atribut URI dari elemen `<Reference>`. Keluaran dari <Transform> terakhir merupakan input untuk algoritma dalam `<DigestMethod>`. Ketika transformasi diterapkan, penandatangan tidak menandatangani dokumen asli, namun dokumen hasil dari transformasi.



Gambar 5. Signature Generation

5.4 Validasi XML Signature

Proses *core generation* dilengkapi dengan proses pembalik yang disebut *core validation*. *Core validation* dispesifikasikan dengan cara yang sama seperti *core generation*, yang juga terdiri dari dua sublangkah: validasi referensi dan validasi tandatangan. Tujuan dari validasi referensi adalah untuk memverifikasi apakah sembarang data yang ditunjuk oleh elemen `<Reference>` tidak berubah. Pemrosesan digest terhadap sumber data dan membuat perbandingan terhadap digest yang tersimpan dalam `<DigestValue>` merupakan pekerjaan proses validasi referensi. Hal yang sama juga menjadi tujuan dari validasi tandatangan yang membandingkan bentuk canonical dari elemen `<SignedInfo>` dengan `<SignatureValue>` yang tersimpan.

Berikut tahapan proses validasi referensi :

1. *canonicalize* elemen `<SignedInfo>` berdasar pada `<CanonicalizationMethod>`.
2. Untuk setiap `<Reference>` dalam `<SignedInfo>`:
    a. ambil objek data untuk diproses *digest*. (Sebagai contoh, aplikasi tandatangan melakukan *dereference* terhadap URI dan menjalankan Transforms yang disediakan oleh penandatangan dalam elemen `<Reference>`, atau mungkin dengan cara lain seperti *local cache*);
    b. lakukan proses penghitungan *digest* dari objek data yang dihasilkan dengan menggunakan algoritma dalam `<DigestMethod>`;

c. bandingkan nilai digest yang dihasilkan dengan nilai dari elemen `<DigestValue>` dalam `<SignedInfo> <Reference>`; jika tidak sama, berarti validasi gagal.

Selanjutnya, berikut tahapan proses validasi tandatangan :

1. ambil informasi kunci dari `<KeyInfo>` atau dari sumber luar lain.
2. Ambil bentuk canonical dari `<SignatureMethod>` dengan menggunakan algoritma yang disebutkan dalam `<CanonicalizationMethod>` dan gunakan hasilnya untuk melakukan konfirmasi `<SignatureValue>` terhadap elemen `<SignedInfo>`.

5.5 Contoh Pemrograman XML Signature

Pada kesempatan ini juga telah dilakukan percobaan untuk pembuatan dan validasi XML Signature dengan memanfaatkan pustaka XML Security yang telah disediakan oleh Apache Foundation dengan project Apache XML Security, yang dapat dikunjungi di http://xml.apache.org/security/dist/. Pada saat percobaan ini dilakukan, penulis menggunakan xml-security versi 1.3.0. Selain itu juga menggunakan JCE provider dari http://www.bouncycastle.org.
Berikut sumber kode program untuk pembuatan XML Signature terhadap dokumen XML seperti pada Kode 1 sebelumnya.

```
import java.io.*;
import java.lang.reflect.*;
import java.security.*;
import java.security.cert.*;
import java.util.*;
import javax.xml.transform.TransformerException;
import org.apache.xpath.XPathAPI;
import org.w3c.dom.*;
import org.apache.xml.security.algorithms.MessageDigestAlgorithm;
import org.apache.xml.security.c14n.*;
import org.apache.xml.security.exceptions.XMLSecurityException;
import org.apache.xml.security.signature.*;
import org.apache.xml.security.keys.*;
import org.apache.xml.security.keys.content.*;
import org.apache.xml.security.keys.content.x509.*;
import org.apache.xml.security.keys.keyresolver.*;
import org.apache.xml.security.keys.storage.*;
import org.apache.xml.security.keys.storage.implementations.*;
import org.apache.xml.security.utils.*;
import org.apache.xml.security.transforms.*;
import org.apache.xml.security.Init;
import  org.apache.xml.serialize.*;

public class SignBuku {
  public static void main(String unused[]) throws Exception {
    org.apache.xml.security.Init.init();

    String fileName = "Buku.xml";
    String signatureFileName = "SignedBuku.xml";
    String keystoreType = "JKS";
```

```
    String keystoreFile = "keystore.jks";
    String keystorePass = "xxxxxx";
    String privateKeyAlias = "mykey";
    String privateKeyPass = "xxxxxx";
    String certificateAlias = "mykey";

    KeyStore ks = KeyStore.getInstance(keystoreType);
    FileInputStream fis = new FileInputStream(keystoreFile);

    ks.load(fis, keystorePass.toCharArray());

    PrivateKey privateKey = (PrivateKey) ks.getKey(privateKeyAlias,

    privateKeyPass.toCharArray());

    javax.xml.parsers.DocumentBuilderFactory dbf =
                    javax.xml.parsers.DocumentBuilderFactory.newInstance();
    dbf.setNamespaceAware(true);
    dbf.setAttribute("http://xml.org/sax/features/namespaces", Boolean.TRUE);
    javax.xml.parsers.DocumentBuilder db = dbf.newDocumentBuilder();
    db.setErrorHandler(new org.apache.xml.security.utils.IgnoreAllErrorHandler());
    org.w3c.dom.Document doc =
               db.parse(new java.io.FileInputStream(new File(fileName)));

    Element bukuElement = null;
    NodeList nodes =
      doc.getElementsByTagNameNS("http://lecturer.ukdw.ac.id/budsus","KoleksiBuku");
    if(nodes != null) {
      bukuElement = (Element)nodes.item(0);
    }

    XMLSignature sig = new XMLSignature(doc,"",XMLSignature.ALGO_ID_SIGNATURE_DSA);
    bukuElement.appendChild(sig.getElement());

    Transforms transforms = new Transforms(doc);
    transforms.addTransform(Transforms.TRANSFORM_ENVELOPED_SIGNATURE);
    transforms.addTransform(Transforms.TRANSFORM_C14N_WITH_COMMENTS);
    sig.addDocument("", transforms,
           org.apache.xml.security.utils.Constants.ALGO_ID_DIGEST_SHA1);

    X509Certificate cert = (X509Certificate) ks.getCertificate(certificateAlias);
    sig.addKeyInfo(cert);
    sig.addKeyInfo(cert.getPublicKey());

    sig.sign(privateKey);
    System.out.println("Penandatanganan selesai");

    FileOutputStream f = new FileOutputStream(new File(signatureFileName));
    XMLUtils.outputDOMc14nWithComments(doc, f);
    f.close();
  }
}
```

Kode 6. SignBuku.java

Program di atas diberinama SignBuku.java yang akan membaca file Buku.xml untuk ditandatangani dengan XML Signature dan disimpan hasilnya dengan nama SignBuku.xml. Kunci yang digunakan pada contoh ini adalah memanfaatkan fasilitas keytool dari J2SE untuk pembuatan kuncinya.

Setelah dikompilasi dan dijalankan, program di atas menghasilkan sebuah file SignBuku.xml dengan isi kurang lebih sebagai berikut :

```
<budsus:KoleksiBuku xmlns:budsus="http://lecturer.ukdw.ac.id/budsus">
   <budsus:buku id="1">
      <budsus:judul>XML Pocket Reference</budsus:judul>
      <budsus:harga>89500</budsus:harga>
   </budsus:buku>
   <budsus:buku id="2">
      <budsus:judul>Windows NT SNMP</budsus:judul>
      <budsus:harga>45</budsus:harga>
   </budsus:buku>
<ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
<ds:SignedInfo>
<ds:CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-
20010315"></ds:CanonicalizationMethod>
<ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#dsa-
sha1"></ds:SignatureMethod>
<ds:Reference URI="">
<ds:Transforms>
<ds:Transform Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature"></ds:Transform>
<ds:Transform Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-
20010315#WithComments"></ds:Transform>
</ds:Transforms>
<ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"></ds:DigestMethod>
<ds:DigestValue>pw60+blJ2y2zWe1Lb6ad1yO4OD0=</ds:DigestValue>
</ds:Reference>
</ds:SignedInfo>
<ds:SignatureValue>CqF9r85bntCsAMK0cpEZ//e5bgsX3ZGDyVtD7X6cQFhZGJQW/vcH6Q==</ds:SignatureValue>
<ds:KeyInfo>
<ds:X509Data>
<ds:X509Certificate>
MIIC8jCCAq8CBEO1ct8wCwYHKoZIzjgEAwUAMF4xCzAJBgNVBAYTAklEMQwwCgYDVQQIEwNESVkx
EzARBgNVBAcTCllvZ3lha2FydGExDTALBgNVBAoTBFVLRFcxCzAJBgNVBAsTAlRJMRAwDgYDVQQD
EwdTdXNhbnRvMB4XDTA1MTIzMDE3NDgxNVoXDTA2MDMzMDE3NDgxNVowXjELMAkGA1UEBhMCSUQx
DDAKBgNVBAgTA0RJWTETMBEGA1UEBxMKWW9neWFrYXJ0YTENMAsGA1UEChMEVUtEVzELMAkGA1UE
CxMCVEkxEDAOBgNVBAMTB1N1c2FudG8wggG4MIIBLAYHKoZIzjgEATCCAR8CgYEA/X9TgR11EilS
30qcLuzk5/YRt1I870QAwx4/gLZRJmlFXUAiUftZPY1Y+r/F9bow9subVWzXgTuAHTRv8mZgt2uZ
UKWkn5/oBHsQIsJPu6nX/rfGG/g7V+fGqKYVDwT7g/bTxR7DAjVUE1oWkTL2dfOuK2HXKu/yIgMZ
ndFIAccCFQCXYFCPFSMLzLKSuYKi64QL8Fgc9QKBgQD34aCF1ps93su8q1w2uFe5eZSvu/o66oL5
V0wLPQeCZ1FZV4661FlP5nEHEIGAtEkWcSPoTCgWE7fPCTKMyKbhPBZ6i1R8jSjgo64eK7OmdZFu
o38L+iE1YvH7YnoBJDvMpPG+qFGQiaiD3+Fa5Z8GkotmXoB7VSVkAUw7/s9JKgOBhQACgYEAwyym
n3JOVIrpyc66qBprG8K9/7iiWWqvKTIS8HN+1Fiz1Koi2/wqAxIkgSktfyy6XH3+r0FIFrzYMADx
Mk44UBYxHq7prPcv0df1V6mrZ8RBMqGE1dy2iMFbtkpKvn6LuNAlbYYN1NKCFwpqfBpDVMnFiV1J
Mm4+B7jfO575qBcwCwYHKoZIzjgEAwUAAzAAMC0CFDuB12OubnMhjQdER7x+phLNN6woAhUAkqzC
gSSnSsG7Wc33hV3BDC6Twqc=
</ds:X509Certificate>
</ds:X509Data>
<ds:KeyValue>
<ds:DSAKeyValue>
<ds:P>
/X9TgR11EilS30qcLuzk5/YRt1I870QAwx4/gLZRJmlFXUAiUftZPY1Y+r/F9bow9subVWzXgTuA
HTRv8mZgt2uZUKWkn5/oBHsQIsJPu6nX/rfGG/g7V+fGqKYVDwT7g/bTxR7DAjVUE1oWkTL2dfOu
K2HXKu/yIgMZndFIAcc=
</ds:P>
<ds:Q>l2BQjxUjC8yykrmCouuEC/BYHPU=</ds:Q>
<ds:G>
9+GghdabPd7LvKtcNrhXuXmUr7v6OuqC+VdMCz0HgmdRWVeOutRZT+ZxBxCBgLRJFnEj6EwoFhO3
zwkyjMim4TwWeotUfI0o4KOuHiuzpnWRbqN/C/ohNWLx+2J6ASQ7zKTxvqhRkImog9/hWuWfBpKL
Zl6Ae1UlZAFMO/7PSSo=
</ds:G>
<ds:Y>
wyymn3JOVIrpyc66qBprG8K9/7iiWWqvKTIS8HN+1Fiz1Koi2/wqAxIkgSktfyy6XH3+r0FIFrzY
MADxMk44UBYxHq7prPcv0df1V6mrZ8RBMqGE1dy2iMFbtkpKvn6LuNAlbYYN1NKCFwpqfBpDVMnF
iV1JMm4+B7jfO575qBc=
</ds:Y>
</ds:DSAKeyValue>
```

```
</ds:KeyValue>
</ds:KeyInfo>
</ds:Signature>
</budsus:KoleksiBuku>
```

Kode 7. Hasil Penandatanganan Buku.xml oleh program SignBuku

Dari dokumen SignBuku.xml tersebut, kita dapat membuat sebuah program yang melakukan validasi terhadap dokumen tersebut dengan menggunakan kunci seperti yang terdapat dalam elemen <ds:KeyInfo>. Kode 8 memperlihatkan kode sumber program yang diberinama VerifiyBuku.java.

```
import java.io.*;
import java.lang.reflect.*;
import java.security.PublicKey;
import java.security.cert.*;
import java.util.*;
import javax.xml.transform.TransformerException;
import org.apache.xpath.XPathAPI;
import org.w3c.dom.*;
import org.apache.xml.security.c14n.*;
import org.apache.xml.security.exceptions.XMLSecurityException;
import org.apache.xml.security.signature.*;
import org.apache.xml.security.keys.*;
import org.apache.xml.security.keys.content.*;
import org.apache.xml.security.keys.content.x509.*;
import org.apache.xml.security.keys.keyresolver.*;
import org.apache.xml.security.keys.storage.*;
import org.apache.xml.security.keys.storage.implementations.*;
import org.apache.xml.security.utils.*;
import org.apache.xml.security.Init;

public class VerifyBuku {
  public static void main(String unused[])  {
    org.apache.xml.security.Init.init();

    String signatureFileName = "SignedBuku.xml";

    javax.xml.parsers.DocumentBuilderFactory dbf =
        javax.xml.parsers.DocumentBuilderFactory.newInstance();

    dbf.setNamespaceAware(true);
    dbf.setAttribute("http://xml.org/sax/features/namespaces", Boolean.TRUE);

    try {
      org.apache.xml.security.Init.init();
      File f = new File(signatureFileName);
      System.out.println("Verifying " + signatureFileName);
      javax.xml.parsers.DocumentBuilder db = dbf.newDocumentBuilder();
      db.setErrorHandler(new org.apache.xml.security.utils.IgnoreAllErrorHandler());
      org.w3c.dom.Document doc = db.parse(new java.io.FileInputStream(f));

      Element sigElement = null;
      NodeList nodes = doc.getElementsByTagNameNS(
      org.apache.xml.security.utils.Constants.SignatureSpecNS,"Signature");

      if(nodes.getLength() != 0) {
        System.out.println("Found " + nodes.getLength() + " Signature elements.");
        sigElement = (Element)nodes.item(0);
```

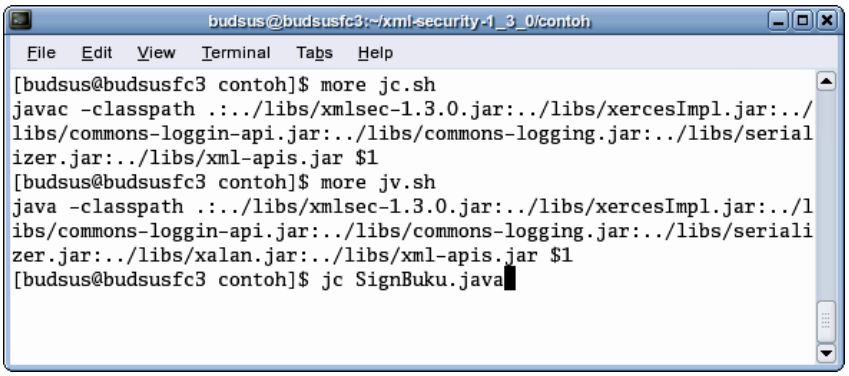
```
        XMLSignature signature = new XMLSignature(sigElement,"");
        KeyInfo ki = signature.getKeyInfo();

        if(ki != null) {
          if(ki.containsX509Data()) {
            System.out.println("Could find a X509Data element in the KeyInfo");
          }
          X509Certificate cert = signature.getKeyInfo().getX509Certificate();

          if(cert != null) {
            System.out.println("The XML signature is " +
                (signature.checkSignatureValue(cert) ? "valid (good)" : "invalid"));
          } else {
            System.out.println("Did not find a Certificate");
            PublicKey pk = signature.getKeyInfo().getPublicKey();
            if(pk != null) {
               System.out.println("The XML signatur is " +
                 (signature.checkSignatureValue(pk) ? "valid (good)" : "invalid"));
            } else {
               System.out.println("Did not find a public key!");
            }
          }
        } else { System.out.println("Did not find a KeyInfo"); }
      }
    }
    catch(Exception e) { e.printStackTrace(); }
  }
}
```

Kode 8. Kode Sumber Program VerifikasiBuku.java

Kedua kode program tersebut diatas merupakan modifikasi dari program yang ditulis oleh Tarak Modi di JavaWorld.com.[2] Secara lengkap pustaka-pustaka yang disertakan saat kompilasi dan eksekusi dapat dilihat pada contoh gambar 6, yang masing-masing tersimpan dalam file shell script jc.sh dan jv.sh.



Gambar 6. Contoh perintah kompilasi dan eksekusi program XML Signature dengan Java

[2] Modi, Tarak, *Safeguard your XML-based messages: Create secure Web services with Apache XML Security*, http://www.javaworld.com/javaworld/jw-12-2002/xmlsecurity/jw-1220-xmlsecurity.zip, Tanggal Akses 28 Desember 2005

6. XML Encryption

Permasalahan utama yang terdapat pada metode pengamanan XML, khususnya yang menerapkan XML *Encryption* ini adalah, bagaimana cara mengimplementasikan teknologi *Cryptography* ke dalam sebuah dokumen XML tanpa merusak struktur dokumen tersebut. Masalah ini timbul disebabkan oleh karena XML merupakan sebuah dokumen data yang terstruktur, sehingga setiap sistem yang mengandalkan pertukaran datanya melalui XML sangat membutuhkan validitas struktur tersebut. Sedangkan di lain pihak, teknologi *Cryptography* memiliki tujuan untuk menyembunyikan segala bentuk informasi ke dalam bentuk data yang tidak terbaca oleh manusia. Oleh karena perbedaan kedua karatkter inilah yang menyebabkan sebuah dokumen XML yang dienkripsi akan kehilangan validitas terhadap struktur yang sudah didefinisikan.

Selain permasalahan di atas, XML *Encryption* juga diharapkan untuk dapat mendukung teknik *Multiple Encryption*[3], yaitu kemampuan untuk mengenkripsi beberapa bagian data dalam dokumen yang sama dengan penanganan yang berbeda. Hal tersebut dibutuhkan oleh sebuah dokumen XML, sebab sebagai teknologi pertukaran data dalam sebuah sistem terdistribusi, dimana data-data yang dibawa dimungkinkan berasal dari beberapa pengguna dana memerlukan pengamanan yang berbeda-beda.

Sebagai standar pertukaran data yang menjembatani berbagai sistem dengan aturan yang berbeda, maka XML *Encryption* berusaha mendukung sebanyak mungkin metode-metode *Cryptography* yang ada. Hal ini dimaksudkan agara tiap pengguna atau pun sistem yang melakukan transaksi dapat menggunakan fasilitas enkripsi yang tersedia dan tidak terbatas pada beberapa metode saja. Dengan kondisi tersebut maka dimungkinkan terdapat lebih dari satu metode yang digunakan dalam sebuah dokumen.

Untuk menerapkan *Cryptography* yang bersifat *multiple* pada sebuah dokumen XML, maka *cryptography* tersebut harus mampu memenuhi kriteria dari sebuah dokumen *markup language*, yaitu terstruktur. Oleh karena itu W3C mengembangkan spesifikasi XML *Encryption*. yang pada dasarnya berupa sebuah elemen standar yang didalamnya memuat parameter-parameter yang digunakan untuk proses enkripsi beserta *chiper text* yang dihasilkan. Dengan menggantikan elemen asli dengan elemen tersebut, diharapkan pengguna dapat tetap mengetahui parameter-parameter yang dibutuhkan saat akan melakukan proses dekripsi pada elemen tersebut.

Spesifikasi XML *Encryption* dapat dilihat pada alamat http://www.w3.org/TR/2002/PR-XMLenc-core-20021003/. Pada bagian ini akan dipaparkan aturan-aturan yang ada pada spesifikasi tersebut beserta contoh pemakaian sintaks XML *Encryption*.

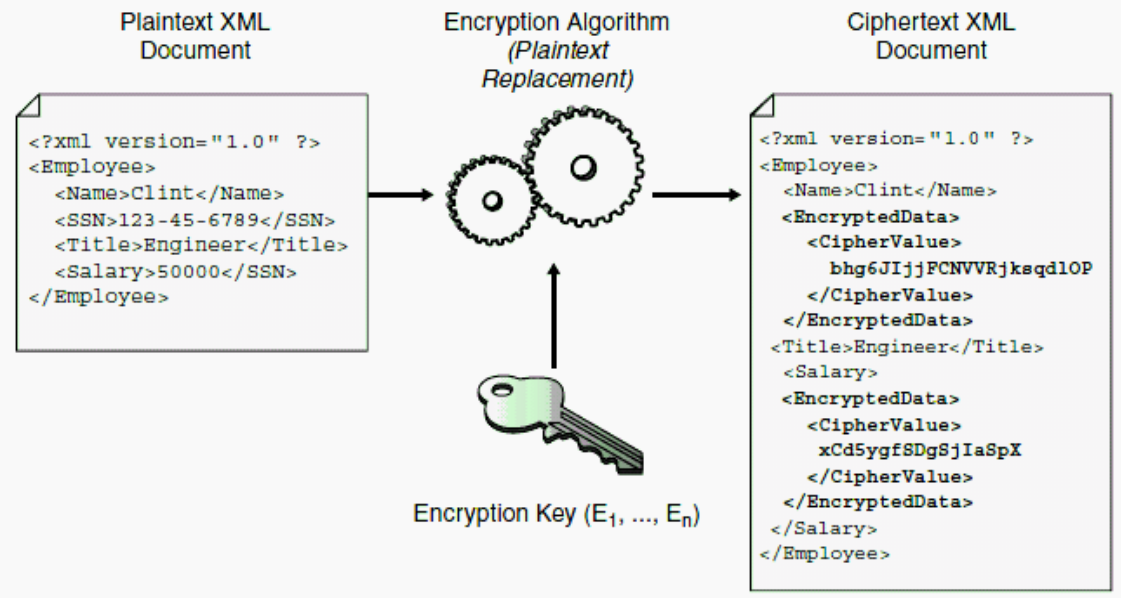
---

[3] Imamura, Takeshi. *XML Encryption Syntax and Processing*, http://www.w3.org/TR/2002/REC-xmlenc-core-20021210/, 2002, Tanggal akses 8-9-2005

6.1 Pilihan Pemakaian XML Encryption

Inti dari XML Encryption adalah elemen `<EncryptedData>` yang didalamnya memuat seluruh informasi mengenai parameter-parameter yang digunakan dalam proses enkripsi. Dalam penggunaannya, elemen tersebut akan menggantikan simpul yang dienkripsi beserta seluruh simpul anak yang dimilikinya. Adapun sebuah proses enkripsi dapat diterapkan pada beberapa macam simpul, antara lain :

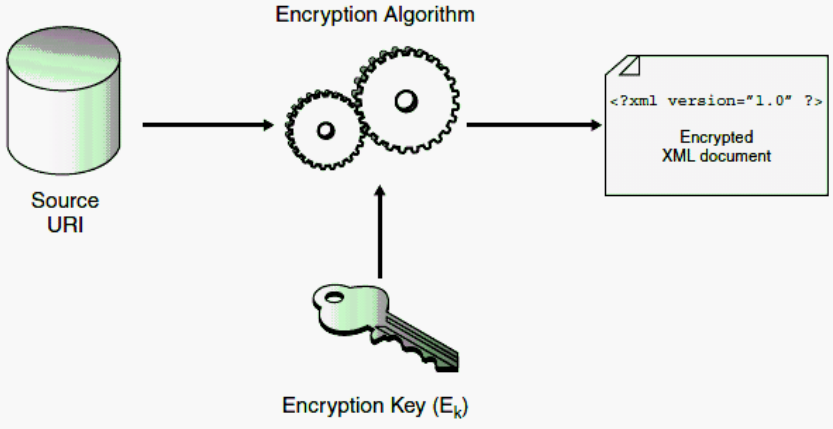
- Elemen beserta *tag* dari elemen tersebut
- Isi dari suatu elemen
- Seluruh isi dari sebuah dokumen XML



Gambar 7. Contoh blok diagram penyandian dengan XML Encryption [4, p. 233]

Seperti yang ditunjukkan pada gambar 7, XML Encryption dapat diterapkan untuk menyandikan baik pada level suatu elemen (tag), dalam contoh adalah elemen <SSN>, dan juga isi dari suatu elemen tertentu, dalam contoh adalah isi dari elemen <Salary>.

Selain melakukan enkripsi pada bagian XML, metode ini juga dapat diterapkan untuk mengenkripsi tipe data lainnya, misalnya gambar atau data *binary* lainnya (lihat gambat 8). Untuk kondisi semacam ini, hasil enkripsi data akan dikonversi ke format base64 sehingga dapat disimpan dalam elemen `<EncryptedData>`. Sedangkan pada proses dekripsi, hasil dekripsi akan langsung dikonversi kembali ke bentuk asal sehingga dapat langsung digunakan oleh pengguna.



Gambar 8. Contoh blok diagram penyandian data oktet [4, p. 229]

Untuk mengidentifikasi elemen-elemen pada spesifikasi XML Encryption, maka telah disediakan sebuah *Namespace* yang dapat digunakan, yaitu xmlns:xenc='http://www.w3.org/2001/04/XMLenc#'. Namespace tersebut juga digunakan untuk mengidentifikasi algoritma yang digunakan pada sebuah proses enkripsi.

Berikut adalah struktur dari elemen `<EncryptedData>` yang dikembangkan oleh W3C dalam spesifikasinya. Pada daftar tersebut dilengkapi dengan keterangan mengenai jumlah pemunculan yang boleh terjadi pada elemen tersebut. Beberapa tanda tersebut antara lain: '?' (boleh tidak muncul atau hanya muncul satu kali), '+' (harus muncul minimal satu kali), dan '*' (boleh tidak muncul atau muncul beberapa kali). Ada pun struktur dari elemen EncryptedData dapat ditunjukkan pada kode 9.

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

Kode 9. Elemen <EncryptedData>

6.1 Sintak XML Encryption

Elemen `<EncryptedData>`

Elemen <EncryptedData> merupakan elemen teratas dari elemen XML Encryption. Di dalamnya terdapat informasi-informasi yang digunakan dalam proses enkripsi. Dalam dokumen XML yang terenkripsi, elemen ini akan menggantikan elemen atau pun isi elemen yang dienkripsi, sebaliknya pada proses dekripsi, elemen ini akan digantikan oleh elemen asli.

Data atau *plain text* yang disimpan didalam element ini tidak selalu berbantuk teks, tetapi juga dapat berupa data *binary*. Untuk memproses data *binary* tentu data tersebut harus dikonversikan terlebih dahulu kedalam bentuk teks yang dapat dibaca, dalam hal ini format yang digunakan adalah format Base64. Untuk membantu proses dekripsi, maka informasi mengenai format *plain text* dicantumkan sebagai nilai atribut MimeType pada element <EncryptedData>.

Untuk proses dekripsi data *binary*, hasilnya tentu saja tidak akan diletakan pada dokumen XML seperti pada proses dekripsi biasa, tatapi hal tersebut tergantung pada kebutuhan sistem yang memanggil proses dekripsi tersebut, sehingga hal tersebut berada diluar ketentuan spesifikasi XML Encryption.

Dari struktur seperti yang tertulis pada Kode 6, terlihat bahwa sebuah elemen <EncryptedData> memiliki beberapa simpul anak, yaitu <EncryptionMethod>, <ds:KeyInfo>, <CipherData>, dan <EncryptionProperties>. Dan elemen pokok yang harus terdapat dalam elemen <EncryptedData> adalah elemen <CipherData>. Elemen ini merupakan elemen yang memuat informasi mengenai data yang telah dienkripsi.

Sedangkan elemen selain <CipherData> merupakan elemen yang bersifat pilihan, artinya elemen tersebut boleh tidak disertakan dalam elemen <EncryptedData>. Jika elemen-elemen tersebut tidak disertakan, maka pengguna bertanggung jawab untuk menyediakan informasi tersebut secara manual jika sistem memerlukannya.

Pengguna diperbolehkan untuk melakukan suatu proses yang dinamakan *Super-Encryption*, yang berarti melakukan enkripsi terhadap sebuah elemen <EncryptedData> lainnya. Dalam hal ini enkripsi harus dilakukan pada seluruh elemen termasuk *tag* pembuka dan penutup.

Sebuah elemen <EncryptedData> tidak diperbolehkan untuk memiliki simpul anak berupa <EncryptedData> lainnya, kecuali jika simpul anak tersebut berupa elemen <EncryptedKey>. Berikut ini merupakan schema definition dari elemen <EncryptedType>.

```
<complexType name='EncryptedType' abstract='true'>
<sequence>
<element name='EncryptionMethod' type='xenc:EncryptionMethodType'
minOccurs='0'/>
<element ref='ds:KeyInfo' minOccurs='0'/>
<element ref='xenc:CipherData'/>
<element ref='xenc:EncryptionProperties' minOccurs='0'/>
</sequence>
<attribute name='Id' type='ID' use='optional'/>
<attribute name='Type' type='anyURI' use='optional'/>
```

```
<attribute name='MimeType' type='string' use='optional'/>
<attribute name='Encoding' type='anyURI' use='optional'/>
</complexType>
```

Sebuah Elemen `<EncryptedType>` memiliki beberapa attribute, antaralain:

- **`Id`**, berfungsi untuk memberikan sebuah tanda berupa teks untuk elemen tertentu. *Attribute* ini sering digunakan pada saat pengguna menggunakan proses *Super-Encryption* atau pun untuk menandai sebuah EncryptedKey agar mudah dikenali.
- **`Type`**, memberikan informasi mengenai tipe dari *plain text* yang telah dienkripsi. *Attribute* ini memiliki dua macam tipe *plain text*, yaitu '*element*' atau 'content'. Informasi ini dibutuhkan agar sistem dapat lebih mudah dalam mengembalikan data hasil proses dekripsi.
  Misal : 'http://www.w3.org/2001/04/XMLenc#Element',
  'http://www.w3.org/2001/04/XMLenc#Content'.
- **`MimeType`**, data teks yang mendeskripsikan format data yang digunakan oleh *plain text*. Misal : 'image/png', 'text/XML'.
- **`Encoding`**, sebuah data berupa URI yang memberikan informasi mengenai metode encoding yang digunakan. Misal : 'http://www.w3.org/2000/09/XMLdsig#base64'.

Elemen `<EncryptionMethod>`

Elemen ini mendefinisikan algoritma yang digunakan pada proses enkripsi. Jika elemen ini tidak disertakan maka pengguna harus mengetahui algoritma yang dipakai. Berikut adalah *schema* dari elemen ini.

```
<complexType name='EncryptionMethodType' mixed='true'>
<sequence>
<element name='KeySize' minOccurs='0' type='xenc:KeySizeType'/>
<element name='OAEPparams' minOccurs='0' type='base64Binary'/>
<any namespace='##other' minOccurs='0' maxOccurs='unbounded'/>
</sequence>
<attribute name='Algorithm' type='anyURI' use='required'/>
</complexType>
```

Nama algoritma yang digunakan disimpan pada *attribute* Algorithm yang nilainya bertipe URI. Misal : 'http://www.w3.org/2001/04/XMLenc#rsa-1_5' menunjuk bahwa algoritma yang digunakan adalah RSA-1.5. Daftar selengkapnya menyangkut beberapa identitas URI algoritma-algoritma yang dapat digunakan dalam XML Encryption dapat dilihat pada tabel 2.

Tabel 2. URI untuk algoritma-algoritma dalam XML EncryptionMethod

Tipe Algoritma	***Nama Algoritma***	***URI***
Block Encryption	3DES (*required*)	http://www.w3.org/2001/04/xmlenc#des3-cbc
	AES-123 (*required*)	http://www.w3.org/2001/04/xmlenc#aes128-cbc

Tipe Algoritma	*Nama Algoritma*	*URI*
	AES-256 (*required*)	http://www.w3.org/2001/04/xmlenc#aes256-cbc
	AES-192 (*optional*)	http://www.w3.org/2001/04/xmlenc#aes192-cbc
Stream Encryption	ARCFOUR (*optional*)	http://www.w3.org/2001/04/xmlenc#arcfour
Key Transport	RSA-v1.5 (*required*)	http://www.w3.org/2001/04/xmlenc#rsa-1_5
	RSA-OAEP (*required*)	http://www.w3.org/2001/04/xmlenc#rsa-oaep-mgf1p
Key Agreement	Diffie-Hellman (*optional*)	http://www.w3.org/2001/04/xmlenc#dh
Symmetric Key Wrap	3DES KeyWrap (*required*)	http://www.w3.org/2001/04/xmlenc#kw-3des
	AES-123 KeyWrap (*required*)	http://www.w3.org/2001/04/xmlenc#kw-aes128
	AES-256 KeyWrap (*required*)	http://www.w3.org/2001/04/xmlenc#kw-aes256
	AES-192 KeyWrap (*optional*)	http://www.w3.org/2001/04/xmlenc#kw-aes192
Message Digest	SHA1	http://www.w3.org/2000/09/xmldsig#sha1
	SHA256	http://www.w3.org/2001/04/xmlenc#sha256
	SHA512	http://www.w3.org/2001/04/xmlenc#sha512
	RIPEMD-160	http://www.w3.org/2001/04/xmlenc#ripemd160

Elemen `<ds:KeyInfo>`

Elemen ini berasal dari spesifikasi XML *Digital Signature* yang berguna untuk menyimpan informasi mengenai kunci yang di gunakan untuk mengekripsi data yang disimpan pada bagian `<CipherData>`. Nama kunci yang digunakan disimpan pada elemen `<ds:KeyName>`.

Elemen `<AgreementMethod>` digunakan jika kunci perlu didistribusikan pada sebuah kelompok tertentu. Elemen ini menyimpan informasi mengenai algoritma yang digunakan serta kunci yang dibutuhkan untuk menciptakan kunci.

XML *Encryption* juga mendukung penggunaan *Session Key* pada proses enkripsinya. Selain itu kunci dari sebuah elemen `<EncryptedData>` dapat diletakan secara terpisah dan dienkripsi dengan sebuah kunci lain. Informasi mengenai kunci yang dipisahkan tersebut disimpan dalam sebuah elemen `<EncryptedKey>` yang dapat diletakan baik sebagai simpul anak dari ds:KeyInfo atau pun pada dokumen XML yang berbeda.

Informasi mengenai lokasi *Session Key* tersebut disimpan dapat dilihat pada elemen `<RetrievalMethod>`. Elemen ini memiliki sebuah atribut URI yang berfungsi menyimpan lokasi dimana elemen `<EncryptedKey>` disimpan.

Elemen <EncryptedKey> juga diturunkan dari element <EncryptedType> yang juga digunakan untuk membentuk elemen <EncryptedData>, sehingga diantara kedua elemen tersebut banyak terdapat kemiripan baik element maupun fungsinya.

Perbedaan mendasar dari keduanya adalah, element <EncryptedData> digunakan untuk menyimpan berbagai format data khususnya berupa teks, sedangkan element <EncryptedKey> hanya digunakan khusus untuk menyimpan data bertipa kunci. Selain itu, sama seperti pada element <EncryptedData>, elemen <EncryptedKey> juga tidak boleh memiliki *node* anak berupa element <EncryptedKey> juga. Berikut shema cari element <EncrypredKey>.

```
<element name='EncryptedKey' type='xenc:EncryptedKeyType'/>
<complexType name='EncryptedKeyType'>
<complexContent>
<extension base='xenc:EncryptedType'>
<sequence>
<element ref='xenc:ReferenceList' minOccurs='0'/>
<element name='CarriedKeyName'   type='string'  minOccurs='0'/>
</sequence>
<attribute name='Recipient' type='string' use='optional'/>
</extension>
</complexContent>
</complexType>
```

Elemen <CipherData>

Elemen ini merupakan elemen pokok yang harus ada dalam sebuah elemen <EncryptedData>. Elemen ini berfungsi untuk menyimpan data yang sudah terenkripsi.

```
<element name='CipherData' type='xenc:CipherDataType'/>
<complexType name='CipherDataType'>
<choice>
<element name='CipherValue' type='base64Binary'/>
<element ref='xenc:CipherReference'/>
</choice>
</complexType>
```

Elemen ini memiliki dua macam simpul anak, yaitu :

- CipherValue, digunakan untuk menyimpan *cipher text* dalam bentuk teks.
- xenc:CipherReference, digunakan jika *cipher text* diletakan secara terpisah pada sebuah dokumen yang dapat diakses melalui alamat URI.

Elemen <EncryptionProperties>

Elemen ini berfungsi untuk menyimpan informasi-informasi yang berhubungan dengan proses enkripsi, seperti : hari, tanggal, nomer seri *hardware* yang digunakan, dan sebagainya.

```
<element name='EncryptionProperties'
         type='xenc:EncryptionPropertiesType'/>
```

```
<complexType name='EncryptionPropertiesType'>
<sequence>
<element ref='xenc:EncryptionProperty' maxOccurs='unbounded'/>
</sequence>
<attribute name='Id' type='ID' use='optional'/>
</complexType>
<element name='EncryptionProperty'
         type='xenc:EncryptionPropertyType'/>
<complexType name='EncryptionPropertyType' mixed='true'>
<choice maxOccurs='unbounded'>
<any namespace='##other' processContents='lax'/>
</choice>
<attribute name='Target' type='anyURI' use='optional'/>
<attribute name='Id' type='ID' use='optional'/>
<anyAttribute namespace="http://www.w3.org/XML/1998/namespace"/>
</complexType>
```

Dari *schema* tersebut, terlihat bahwa elemen `<EncryptionProperties>` terdiri dari sejumlah elemen `<EncryptionProperty>`.

6.2 Aturan Pemrosesan XML Encryption

Pada draft spesifikasi XML Encryption, disebutkan ada 3 jenis aturan dalam pemrosesan sebuah XML Encryption, yaitu aplikasi, penyandi (*encryptor*) dan pembongkar sandi (*decryptor*).

Aplikasi

Istilah aplikasi menunjuk pada sembarang entitas yang dipercaya dalam penerapan XML Encryption. Beberapa hal tidak tercakup dalam XML Encryption dan harus ditangani oleh aplikasi, misalnya validasi plaintext (setelah didekripsi) dan serialisasi (koversi dari XML ke string oktet).

Encryptor (Penyandi)

Istilah ini menunjuk pada suatu entitas pemrosesan yang melakukan operasi enkripsi. Encryptor berperan untuk membangkitkan elemen `<EncryptedData>`, baik untuk sebuah kunci atau data. Encryptor bertanggung jawab untuk membangkitkan dan menggabungkan semua elemen XML Encryption termasuk kunci, data terkodekan, dan semua atribut. Selain itu juga bertanggung jawab dalam hal penyimpanan kunci dan struktur selama proses berlangsung.

Dalam spesifikasi XML Encryption, didefinisikan pula algoritma proses enkripsi sebagai berikut:

1. Pilihlah algoritma (dan parameter) yang digunakan dalam penyandian data.
2. Hasilkan atau ambil kunci enkripsi yang digunakan.
3. Lokasikan deretan oktet untuk dienkrip.
    a. Jika data yang dienkrip adalah sebuah elemen XML atau isi dari elemen XML, deretan oktet adalah string terkodekan dengan UTF-8 atau isi dari elemen itu sendiri. Deretan oktet UTF-8 ini disandikan dengan kunci yang diperoleh pada tahap sebelumnya.

Aplikasi enkripsi direkomendasikan untuk menggunakan atribut pilihan `Type` dari elemen EncrypredData dengan nilai yang sesuai untuk mendukung proses pengembalian data pada proses dekripsi.

b. Jika data yang dienkrip adalah deretan oktet eksternal, sandikan dengan kunci yang diperoleh pada tahap sebelumnya.

4. Bangun struktur XML untuk langkap enkripsi ini
   a. Jika data yang telah dienkrip adalah elemen XML atau isi dari elemen XML, data yang tidak tersandi dibuang dan diganti dengan struktur XML baru dengan menggunakan pengkodean yang sama seperti dokumen XML induknya.
   b. Jika data yang dienkrip adalah deretan oktet eksternal, buat struktur `<EncryptedData>` dengan memasukkan atau mereferensikan data tersandi dan gunakan elemen tersebut sebagai elemen level teratas dalam sebuah dokumen XML baru atau sisipkan ke dalam dokumen XML lain (proses ini tergantung dari aplikasi).

Decryptor (Pembongkar Sandi)

Decryptor menunjuk pada entitas yang melakukan proses kebalikan dari encryptor. Decryptor bertanggung jawab untuk penguraian dan dekripsi terhadap sembarang paket `<EncryptedData>` yang ada dalam dokumen XML. Selain itu juga bertanggung jawab untuk melakukan proses decoding dan dekripsi sembarang data tersandi kepada aplikasi untuk diproses selanjutnya.

Dalam spesifikasi XML Encryption, didefinisikan pula algoritma proses dekripsi (baik untuk elemen `<EncryptedData>` atau `<EncryptedKey>`) sebagai berikut:

1. Parsing elemen untuk menentukan algoritma, paramter dan kunci yang digunakan.
2. Jika kunci enkripsi data dalam bentuk terenkrip, lokasikan kunci terkait untuk mendekripsinya. (mungkin akan terjadi proses rekursi atau membaca dari penyimpan lokal menggunakan atribut yang disebutkan)
3. Dekrip data yang ada dalam elemen `<ChiperData>`. Jika data adalah XML, deretan oktet yang dihasilkan diterjemahkan sebagai string karakter XML yang terkodekan UTF-8 yang menyatakan elemen atau data dari elemen XML.
   a. Jika `<ChiperData>` berisi elemen `<ChiperValue>`, ambil deretan oktet dengan melakukan proses *decoding* base64 terhadap isinya.
   b. Jika `<ChiperData>` berisi sebuah elemen `<ChiperReference>`, lakukan *dereference* terhadap nilai atribut `URI` dan lakukan proses transformasi untuk mendapatkan deretan data oktetnya.
4. Jika hasilnya merupakan struktur `<EncryptedData>` dan atribut `Type` adalah “Element” atau “Content”, maka lakukan pengkodean terhadap hasil dekripsinya seusai dengan pengkodean

dokumen XML induk yang diperlukan. Selain itu, deretan oktet yang dihasilkan adalah hasil akhir.

6.3 Contoh Pemrograman XML Encryption

Dengan memanfaatkan pustaka yang sama untuk percobaan XML Signature, berikut sumber kode program untuk enkripsi dokumen XML seperti pada kode 1, khususnya untuk data elemen <budsus:harga>. Kode 10 merupakan program Encrypter.java yang merupakan modifikasi dari contoh program dari Apache XML Security 1.3.0.

```
/* Program ini merupakan modifikasi sample Encrypter.java dari
 * Apache XML-Security 1.3.0
 * @author Vishal Mahajan (Sun Microsystems)
 * dimodifikasi oleh Budi Susanto
 */
import java.io.File;
import java.io.FileOutputStream;
import java.security.Key;
import javax.crypto.SecretKey;
import javax.crypto.KeyGenerator;
import org.apache.xpath.XPathAPI;
import org.apache.xml.security.keys.KeyInfo;
import org.apache.xml.security.encryption.XMLCipher;
import org.apache.xml.security.encryption.EncryptedData;
import org.apache.xml.security.encryption.EncryptedKey;
import org.apache.xml.security.utils.Constants;
import org.w3c.dom.*;
import javax.xml.transform.TransformerFactory;
import javax.xml.transform.Transformer;
import javax.xml.transform.dom.DOMSource;
import javax.xml.transform.stream.StreamResult;
import javax.xml.transform.OutputKeys;

public class Encrypter {
  // inisialisasi pustaka Apache XML Security
  static { org.apache.xml.security.Init.init(); }

  /* method untuk membaca dan melakukan parsing dokumen XML
   */
  private static Document parseFile(String fileName) throws Exception {
    javax.xml.parsers.DocumentBuilderFactory dbf =
        javax.xml.parsers.DocumentBuilderFactory.newInstance();
    dbf.setNamespaceAware(true);
    javax.xml.parsers.DocumentBuilder db = dbf.newDocumentBuilder();
    Document document = db.parse(fileName);

    return document;
  }

  /* Method untuk membangkitkan kunci untuk menyandikan data
   * Kunci yang terbentuk disimpan atau dikirimkan
   */
  private static SecretKey GenerateAndStoreKeyEncryptionKey()
      throws Exception {
    String jceAlgorithmName = "DESede";
    KeyGenerator keyGenerator = KeyGenerator.getInstance(jceAlgorithmName);
```

```
  SecretKey kek = keyGenerator.generateKey();

  byte[] keyBytes = kek.getEncoded();
  File kekFile = new File("kek");
  FileOutputStream f = new FileOutputStream(kekFile);
  f.write(keyBytes);
  f.close();

  return kek;
}

/* Method untuk membangkitkan kunci symmetric
 */
private static SecretKey GenerateDataEncryptionKey() throws Exception {
  String jceAlgorithmName = "AES";
  KeyGenerator keyGenerator = KeyGenerator.getInstance(jceAlgorithmName);
  keyGenerator.init(128);
  return keyGenerator.generateKey();
}

/* Method untuk menyimpan hasil XML Encryption ke file
 */
private static void outputDocToFile(Document doc, String fileName)
     throws Exception {
  File encryptionFile = new File(fileName);
  FileOutputStream f = new FileOutputStream(encryptionFile);

  TransformerFactory factory = TransformerFactory.newInstance();
  Transformer transformer = factory.newTransformer();
  transformer.setOutputProperty(OutputKeys.OMIT_XML_DECLARATION, "yes");
  DOMSource source = new DOMSource(doc);
  StreamResult result = new StreamResult(f);
  transformer.transform(source, result);

  f.close();
}

public static void main(String doc[]) throws Exception {
  /* Baca dan parsing dokumen XML yang akan disandikan
   */
  Document document = parseFile(doc[0]);

  /* Buat kunci yang digunakan untuk enkripsi elemen (AES)
   */
  Key symmetricKey = GenerateDataEncryptionKey();

  /* Buat kunci untuk enkripsi kunci simmetric (TRIPLEDES)
   */
  Key kek = GenerateAndStoreKeyEncryptionKey();

  String algorithmURI = XMLCipher.TRIPLEDES_KeyWrap;

  /* tentukan algoritma TripleDES untuk enkripsi kunci
   * set mode ke WRAP MODE yang akan mengubah kunci ke array byte
   * agar dapat dikirimkan secara aman dari satu tempat ke lain (sistem file)
   */
  XMLCipher keyCipher = XMLCipher.getInstance(algorithmURI);
  keyCipher.init(XMLCipher.WRAP_MODE, kek);
  EncryptedKey encryptedKey = keyCipher.encryptKey(document, symmetricKey);

  /* Sandikan semua elemen <budsus:harga>
```

```
     */
    Element rootElement = document.getDocumentElement();
    Element elementToEncrypt = null;
    NodeList nodes =
      document.getElementsByTagNameNS("http://lecturer.ukdw.ac.id/budsus","harga");

    algorithmURI = XMLCipher.AES_128;
    for(int i=0; i < nodes.getLength(); i++) {
      elementToEncrypt = (Element)nodes.item(i);
      if (elementToEncrypt != null) {
        XMLCipher xmlCipher = XMLCipher.getInstance(algorithmURI);
        xmlCipher.init(XMLCipher.ENCRYPT_MODE, symmetricKey);

        /* siapkan keyinfo dalam data tersandi
         */
        EncryptedData encryptedData = xmlCipher.getEncryptedData();
        KeyInfo keyInfo = new KeyInfo(document);
        keyInfo.add(encryptedKey);
        encryptedData.setKeyInfo(keyInfo);

        /* doFinal-melakukan enkripsi terhadap isi elemen, bukan elemen itu sendiri
         * doFinal juga mengubah dokumen dengan menempatkan elemen EncrypteData
         * untuk data yang disandikan
         */
        xmlCipher.doFinal(document, elementToEncrypt, true);
      }
    }

    /* simpan ke file hasil enkripsi
     */
    outputDocToFile(document, "encryptedBuku.xml");
  }
}
```

Kode 10. Sumber kode program Encrypter.java

Dari program Encrypter.java kemudian dikompilasi dengan perintah berikut ini :

```
javac -classpath .:../libs/xmlsec-1.3.0.jar:../libs/xercesImpl.jar:../libs/commons-
loggin-api.jar:../libs/commons-logging.jar:../libs/serializer.jar:../libs/xml-
apis.jar $1
```

Setelah itu dijalankan dengan perintah :

```
java -classpath .:../libs/xmlsec-1.3.0.jar:../libs/xercesImpl.jar:../libs/commons-
loggin-api.jar:../libs/commons-
logging.jar:../libs/serializer.jar:../libs/xalan.jar:../libs/xml-apis.jar $1
```

Contoh hasil dari eksekusi program Encrypter.java yang melakukan enkripsi terhadap dokumen Buku.xml dengan XML Encryption dapat dilihat pada Kode 11.

```
<budsus:KoleksiBuku xmlns:budsus="http://lecturer.ukdw.ac.id/budsus">
   <budsus:buku id="1">
      <budsus:judul>XML Pocket Reference</budsus:judul>
      <budsus:harga><xenc:EncryptedData
Type="http://www.w3.org/2001/04/xmlenc#Content"
xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"><xenc:EncryptionMethod
Algorithm="http://www.w3.org/2001/04/xmlenc#aes128-cbc"
```

```
xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"/><ds:KeyInfo
xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
<xenc:EncryptedKey
xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"><xenc:EncryptionMethod
Algorithm="http://www.w3.org/2001/04/xmlenc#kw-tripledes"
xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"/><xenc:CipherData
xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"><xenc:CipherValue
xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">lREtgLxIm9aaVKWwHT3JYJJzh6uU4c76znn4O
eemd/0=</xenc:CipherValue></xenc:CipherData></xenc:EncryptedKey></ds:KeyInfo><xenc:C
ipherData xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"><xenc:CipherValue
xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">NAurgt+v9Qj81KS46JZaSOf0M1jZ/Rk80AC7M
+JCB6Q=</xenc:CipherValue></xenc:CipherData></xenc:EncryptedData></budsus:harga>
   </budsus:buku>
   <budsus:buku id="2">
      <budsus:judul>Windows NT SNMP</budsus:judul>
      <budsus:harga><xenc:EncryptedData
Type="http://www.w3.org/2001/04/xmlenc#Content"
xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"><xenc:EncryptionMethod
Algorithm="http://www.w3.org/2001/04/xmlenc#aes128-cbc"
xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"/><ds:KeyInfo
xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
<xenc:EncryptedKey
xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"><xenc:EncryptionMethod
Algorithm="http://www.w3.org/2001/04/xmlenc#kw-tripledes"
xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"/><xenc:CipherData
xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"><xenc:CipherValue
xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">lREtgLxIm9aaVKWwHT3JYJJzh6uU4c76znn4O
eemd/0=</xenc:CipherValue></xenc:CipherData></xenc:EncryptedKey></ds:KeyInfo><xenc:C
ipherData xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"><xenc:CipherValue
xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">Z3ixndwNBu5dfUaGHLdI0NSNgb+we8n+XLNc6
fwsuCo=</xenc:CipherValue></xenc:CipherData></xenc:EncryptedData></budsus:harga>
   </budsus:buku>
</budsus:KoleksiBuku>
```

Kode 11. Contoh hasil XML Encryption (encryptedBuku.xml) terhadap dokumen Buku.xml

Kode 12 merupakan contoh program yang melakukan dekripsi terhadap dokumen XML Encryption encryptedBuku.xml.

```
/* Program ini merupakan modifikasi sample Encrypter.java dari
 * Apache XML-Security 1.3.0
 * @author Vishal Mahajan (Sun Microsystems)
 * dimodifikasi oleh Budi Susanto
 */
import java.io.File;
import java.io.FileOutputStream;

import java.security.Key;
import javax.crypto.SecretKey;
import javax.crypto.SecretKeyFactory;
import javax.crypto.spec.DESedeKeySpec;

import org.apache.xml.security.encryption.XMLCipher;
import org.apache.xml.security.utils.JavaUtils;
import org.apache.xml.security.utils.EncryptionConstants;

import org.w3c.dom.*;
```

```
import javax.xml.transform.TransformerFactory;
import javax.xml.transform.Transformer;
import javax.xml.transform.dom.DOMSource;
import javax.xml.transform.stream.StreamResult;
import javax.xml.transform.OutputKeys;

public class Decrypter {
  static { org.apache.xml.security.Init.init(); }

  /* method untuk membaca dan mengubah ke objek Document
   * terhadap file XML terenkripsi yang akan didekrip
   */
  private static Document loadEncryptionDocument(String fileName) throws Exception {
    File encryptionFile = new File(fileName);
    javax.xml.parsers.DocumentBuilderFactory dbf =
            javax.xml.parsers.DocumentBuilderFactory.newInstance();
    dbf.setNamespaceAware(true);
    javax.xml.parsers.DocumentBuilder db = dbf.newDocumentBuilder();
    Document document = db.parse(encryptionFile);

    return document;
  }

  /* Method untuk membaca kunci yang tersimpan sebelumnya
   * untuk mendekrip kunci terenkrip
   */
  private static SecretKey loadKeyEncryptionKey() throws Exception {
    String fileName = "kek";
    String jceAlgorithmName = "DESede";

    File kekFile = new File(fileName);

    DESedeKeySpec keySpec =
       new DESedeKeySpec(JavaUtils.getBytesFromFile(fileName));
    SecretKeyFactory skf =
       SecretKeyFactory.getInstance(jceAlgorithmName);
    SecretKey key = skf.generateSecret(keySpec);

    return key;
  }

  /* Method untuk menuliskan hasil dekripsi ke file
   */
  private static void outputDocToFile(Document doc, String fileName)
        throws Exception {
    File encryptionFile = new File(fileName);
    FileOutputStream f = new FileOutputStream(encryptionFile);

    TransformerFactory factory = TransformerFactory.newInstance();
    Transformer transformer = factory.newTransformer();
    transformer.setOutputProperty(OutputKeys.OMIT_XML_DECLARATION, "yes");
    DOMSource source = new DOMSource(doc);
    StreamResult result = new StreamResult(f);
    transformer.transform(source, result);

    f.close();
  }

  public static void main(String param[]) throws Exception {
    // baca dokumen XML yang akan didekrip
    Document document = loadEncryptionDocument(param[0]);
```

```
    /* Ambil kunci yang digunakan untuk melakukan dekripsi data kunci tersandi
     */
    Key kek = loadKeyEncryptionKey();

    /* buat dan inisialisasi sebuah ciper dengan kunci data-decryption
     * Kunci digunakan untuk dekripsi data XML yang akan diambil
     * dari keyinfo dalam EncrypteData menggunakan kek.
     */
    XMLCipher xmlCipher = XMLCipher.getInstance();
    xmlCipher.init(XMLCipher.DECRYPT_MODE, null);
    xmlCipher.setKEK(kek);

    /* ambil elemen data yang tersandi dari dokumen DOM
     */
    String nsURI = EncryptionConstants.EncryptionSpecNS;
    String localName = EncryptionConstants._TAG_ENCRYPTEDDATA;
    NodeList nodes = document.getElementsByTagNameNS(nsURI, localName);
      for (int i=0; i < nodes.getLength(); i++) {
        Element encryptedDataElement = (Element)nodes.item(i);
        if (encryptedDataElement != null) {
          /* lakukan proses dekripsi
           */
          xmlCipher.doFinal(document, encryptedDataElement);
        }
      }

    /* tulis hasil dekripsi ke file
     */
    outputDocToFile(document, "decryptedBuku.xml");
  }
}
```

7. Daftar Pustaka

[1] Arciniegas A., Fabio, "*XML Developer's Guide*", McGrawHill Companies, Inc., 2001

[2] Bartel, Mark, etc, " *XML-Signature Syntax and Processing*", W3C Recommendation 12 February 2002, http://www.w3.org/TR/2002/REC-xmldsig-core-20020212/, tanggal akses 05-12-2005

[3] Bosworth, Seymour; Kabay, M.E., "*Computer Security Handbook*", 4th edition, John Wiley & Sons, Inc., 2002

[4] Daurnaee, Blake, "*XML Security*", McGrawHIll Companies, Inc., 2002

[5] Hirsch, Frederick, "Getting Started with XML Security", 28 November 2002, http://www.sitepoint.com/subcat/xml, tanggal akses 05-12-2005

[6] Imamura, Takeshi, etc., "*XML Encryption Syntax and Processing*", WG Working Draft 26 June 2001, http://www.w3.org/TR/2002/REC-xmlenc-core-20021210/, tanggal akses 05-12-2005

[7] Ray Djajadinata. "*XML Encryption keeps your XML documents safe and secure*", http://www.javaworld.com/javaworld/jw-08-2002/jw-0823-securexml.html, JavaWorld.com, 2003, tanggal akses 05-12-2005

[8] Reagle, Joseph. "*XML Encryption Requirements*", http://www.w3.org/TR/2002/NOTE-xml-encryption-req-20020304, 2002, tanggal akses 05-12-2005

[9] Simon, Ed; Madsen, Paul; Adams, Carlisle, "*An Introduction to XML Digital Signatures*", http://www.xml.com/pub/a/2001/08/08/xmldsig.html, Tanggal akses 07-12-2005