# Tanda Tangan Digital Menggunakan XML

Yudha Adiprabowo – NIM : 13506050

*Program Studi Teknik Informatika, Institut Teknologi Bandung*
*Jl. Ganesha 10, Bandung*
E-mail : if16050@students.if.itb.ac.id

## Abstrak

Makalah ini membahas tentang tanda tangan digital untuk mengamankan data-data yang menggunakan struktur XML, seperti sebuah situs web atau data-data yang dikirim melalui protokol-protokol web.

Tanda tangan digital menggunakan XML memiliki beberapa standard yang harus dipatuhi dan dispesifikasikan dalam *The XML-Signature Syntax and Processing specification* / XML DSIG. Dalam melakukan proses penandatanganan suatu dokumen XML harus dilakukan proses *canonicalization* yang suatu proses untuk mengkonversi data yang memiliki lebih dari satu perwakilan menjadi "standar" kanonik perwakilan.

***Kata kunci**: XML*, tanda tangan digital, *canonicalization*, *XML DSIG*.

## 1. Pendahuluan

Tanda tangan digital penting karena memberikan jaminan integritas pesan dari titik ke titik, dan juga dapat memberikan informasi mengenai otentikasi pembuat pesan. Agar lebih efektif, tanda tangan harus menjadi bagian dari data aplikasi, sehingga tanda tangan tersebut dihasilkan pada saat pesan dibuat, dan dapat diverifikasi pada saat pesan akhirnya diterima dan diproses.

SSL / TLS juga menyediakan integritas pesan (serta privasi akan pesan), tetapi hanya menyediakannya saat pesan sedang transit. Setelah pesan diterima oleh server (atau, lebih umumnya, penerima), perlindungan SSL harus "dilucuti" supaya pesan dapat diproses.

Untuk lebih jelasnya, SSL hanya bekerja antara titik-titik akhir komunikasi. Jika saya mengembangkan layanan Web baru dan konvensional menggunakan server HTTP (seperti Apache atau IIS) sebagai *gateway*, atau jika saya berkomunikasi dengan perusahaan besar yang memiliki *SSL accelerators*, integritas pesan hanya terjamin sampai sambungan SSL diakhiri.

Sebagai analogi, lihat sebuah surat konvensional. Jika saya mengirimkan sebuah cek ke perusahaan telepon, saya menandatangani ceknya (pesan) dan menaruhnya dalam amplop untuk mendapatkan privasi dan melakukan pengiriman. Setelah menerima surat, perusahaan telepon membuka amplop, membuangnya, kemudian melakukan pemrosesan terhadap cek tersebut. Saya dapat membuat pesan saya menjadi bagian dari amplop, seperti mengelem tanda pembayaran ke kartu pos dan mengirimkannya, tetapi itu perbuatan bodoh.

Sebuah tanda tangan XML akan menetapkan sejumlah elemen XML yang dapat tertanam dalam, atau terkait dengan, setiap dokumen XML. Ini akan memungkinkan penerima untuk melakukan verifikasi bahwa pesan belum diubah dari apa yang dimaksudkan pengirim.

Spesifikasi sintaks dan pemrosesan tanda tangan XML (*The XML-Signature Syntax and Processing specification* / XML DSIG) merupakan upaya bersama dari W3C dan IETF. Hal ini sudah menjadi rekomendasi W3C sejak Februari 2002. Banyak implementasi tersedia; dalam .NET Framework, terdapat System.Security.Cryptography.Xml. Di dalam *WS-Security triad* -otentikasi, integritas konten, dan privasi konten- XML DSIG menyediakan integritas dan dapat digunakan untuk menyediakan otentikasi pengirim.

## 2. Tanda Tangan Digital dengan Kriptografi

Sebelum kita dapat benar-benar memahami XML DSIG, kita perlu memiliki pemahaman

dasar kriptografi. Saya akan membahas konsep-konsepnya pada bagian ini.

Tanda tangan digital memberikan pemeriksaan integritas pada beberapa konten. Jika satu byte pada pesan yang asli telah dimodifikasi – tambahan angka nol, "2" diubah menjadi "4", atau "Tidak" untuk "Ya", dan seterusnya- maka tanda tangan akan gagal untuk diverifikasi. Berikut adalah cara kerjanya.

Langkah pertama adalah untuk melakukan "hash" pada pesan. Sebuah "hash" kriptografi mengambil *stream* byte dan melakukan konversi menjadi sebuah satu-ukuran nilai yang dikenal sebagai *digest*. *Digest* merupakan sebuah proses satu-arah. Secara komputasi, mustahil membuat kembali pesan dari "hash"-nya, atau mendapatkan dua pesan yang berbeda menghasilkan nilai digest yang sama.

Mekanisme hash yang paling umum adalah mekanisme SHA1, *Secure Hash Algoritm*. Mekanisme ini dibuat oleh Pemerintah Amerika Serikat dan dirilis sebagai standar pada tahun 1995; spesifikasi lengkap tersedia di http://www.itl.nist.gov/fipspubs/fip180-1.htm.
SHA1 mengambil pesan apapun hingga sepanjang $2^{64}$ byte dan menghasilkan 20-byte. (Berarti terdapat $2^{160}$ kemungkinan nilai *digest*; untuk perbandingan, saat ini perkiraan jumlah proton di alam semesta sekitar $2^{250}$).

Jadi jika saya menghasilkan pesan *M*, dan membuat *digest*, (ditulis sebagai *H(M)*, untuk "hash dari *M*"), dan menerima *M* dan *H*(*M*), Anda dapat membuat sendiri *digest H*'(*M*), dan jika kedua nilai *digest* cocok, kita tahu bahwa Anda mendapatkan apa yang saya kirim. Untuk melindungi *M* dari perubahan, saya hanya perlu melindungi *H*(*M*) agar tidak diubah.

Ada dua pendekatan umum untuk melakukan hal di atas. Yang pertama adalah melakukan campuran bersama rahasia ke digest. Dengan kata lain, membuat *H*(*S*+*M*). Bila Anda menerima pesan, Anda gunakan sendiri salinan *S* untuk membuat *H*'(*S*+*M*). *Digest* baru ini adalah yang disebut HMAC, atau *Hashed Message Authentication Code*.

Apabila kita menggunakan HMAC, kekuatan perlindungan integritas tergantung pada kemampuan penyerang untuk mengetahui *S*. Oleh karena itu, *S* harus sesuatu yang tidak mudah ditebak, dan sesuatu yang harus sering berubah. Salah satu cara terbaik untuk memenuhi persyaratan tersebut adalah menggunakan Kerberos. Dalam Kerberos, penguasa pusat mendistribusikan "tiket" yang berisi kunci sesi sementara kapanpun kedua entitas ingin berkomunikasi. Sesi ini akan digunakan sebagai rahasia bersama. Ketika saya ingin mengirimkan Anda tanda tangan, saya mendapatkan tiket untuk berbicara dengan Anda. Saya buka bagian dari saya untuk mendapatkan tiket S, dan mengirim pesan Anda, HMAC-nya, dan bagian dari tiket. Anda membuka tiket (menggunakan password saat Anda mendaftar dengan Kerberos) dan mendapatkan *S* dan informasi tentang identitas saya. Sekarang Anda dapat mengambil pesan, *M*, menghasilkan *H*'(*S*+*M*), dan lihat apakah mereka cocok. Jika mereka cocok, Anda tahu bahwa Anda menerima pesan saya utuh, dan Kerberos memberitahukan anda siapa saya.

Cara lain untuk melindungi digest adalah dengan menggunakan kriptografi kunci publik, seperti RSA. Dalam kriptografi kunci publik, terdapat dua kunci, sebuah kunci privat, hanya dikenal dengan pemiliknya, dan sebuah kunci publik, dapat diakses oleh siapa saja yang ingin berkomunikasi dengan pemilik kunci. Dalam kriptografi kunci publik, sesuatu yang dienkripsi dengan kunci pribadi dapat didekripsi dengan kunci publik, dan sebaliknya.

Menggunakan RSA, saya menghasilkan digest, *H*(*M*), dan mengenkripsi dengan kunci privat saya, {H(M)} kunci privat, yang merupakan tanda tangannya. Bila Anda menerima pesan, *M*, Anda menghasilkan digest, *H*'(*M*), dan mendekripsi tanda tangan menggunakan kunci publik, mendapatkan *H*(*M*) yang dihasilkan. Jika *H*(*M*) dan *H*'(*M*) adalah sama, maka kita tahu bahwa *M* adalah sama. Selanjutnya, Anda tahu bahwa saya adalah pengirim pesan.

**3. Format Tanda Tangan XML**

Struktur dasar dari sebuah tanda tangan XML adalah sebagai berikut.

```
<Signature>
  <SignedInfo>
    <SignatureMethod />
    <CanonicalizationMethod />
      <Reference>
        <Transforms>
        <DigestMethod>
        <DigestValue>
      </Reference>
```

```
        <Reference /> dst.
    </SignedInfo>
    <SignatureValue />
    <KeyInfo />
    <Object />
</Signature>
```

*SignedInfo* mengandung atau mereferensikan data yang ditandatangani dan menspesifikasikan algoritma apa yang digunakan. Elemen *CanonicalizationMethod* dan *SignatureMethod* digunakan oleh elemen *SignatureValue* dan disertakan dalam *SignedInfo* untuk melindungi mereka dari gangguan. Satu atau lebih elemen *Reference* menentukan sumber daya yang ditandatangani dan perubahan apapun yang akan diterapkan pada pesan awal sebelum penandatanganan. *DigestMethod* menentukan algoritma hash sebelum menerapkan hash. *DigestValue* berisi hasil penerapan algoritma hash ke pesan awal.

*SignatureValue* adalah elemen yang berisi hasil *encode* tanda tangan dalam Base64 -tanda tangan yang dihasilkan dengan parameter yang ditentukan dalam elemen *SignatureMethod*- dari *SignedInfo* setelah penerapan algoritma sesuai dengan yang dispesifikasikan oleh *CanonicalizationMethod*.

*KeyInfo* adalah elemen yang secara opsional memungkinkan pengirim untuk memberikan penerima kunci yang memvalidasi tanda tangan, biasanya dalam bentuk satu atau lebih sertifikat digital X.509. Pihak yang menerima harus mengidentifikasi kunci dari konteks jika *KeyInfo* tidak ada.

Elemen *Object* (opsional) berisi data yang ditandatangani jika tanda tangan ini bersifat *enveloping* (membungkus) data awal.

Bila hendak memvalidasi tanda tangan XML, sebuah prosedur yang disebut *Core Validation* harus diikuti.

- Referensi Validasi: Setiap *digest* milik referensi diverifikasi oleh mengambil pesa awal dan menerapkan setiap perubahan dan kemudian dilakukan pembuatan *digest*-nya. Hasilnya dibandingkan dengan yang tertera pada *DigestValue*, jika mereka tidak cocok, validasi gagal.
- Validasi Tanda Tangan: elemen *SignedInfo* diserialisasi dengan menggunakan metode *canonicalization* yang ditentukan dalam *CanonicalizationMethod*, kunci data diambil dari *KeyInfo* atau dengan cara lain, dan tanda tangan diverifikasi menggunakan metode yang ditetapkan dalam *SignatureMethod.*

Prosedur ini menetapkan apakah pesan awal benar-benar ditandatangani oleh pihak yang diduga. Namun, karena kemungkinan diperpanjang dari *canonicalization* dan metode transformasi metode, pihak yang memverifikasi juga harus memastikan bahwa apa yang sebenarnya ditandatangani atau dibuat *digest*-nya benar-benar ada dalam data yang asli, dengan kata lain, apakah algoritma yang digunakan dapat dipercaya untuk tidak mengubah maksud data.

## 4. Saran dan Kritik pada Tanda Tangan XML

Ada kritik yang ditujukan pada keharusan untuk melakukan *canonicalization* XML untuk menandatangani dan enkripsi data XML karena kompleksitas, karakteristik dan kinerja yang buruk. Argumen yang ada adalah, melakukan *canonicalization* XML menyebabkan *latency* berlebihan yang cukup besar untuk menjalankan transaksi.

*Canonicalization* (disingkat c14n, dimana 14 merupakan jumlah huruf antara C dan N) adalah suatu proses untuk mengkonversi data yang memiliki lebih dari satu perwakilan menjadi "standar" kanonik perwakilan. Contoh yang paling mendasar adalah tag <container> dan < container > dianggap sama dan memiliki *canonical* yang sama.

Mengapa proses *canonicalization* dianggap membuat proses penandatanganan menjadi lebih rumit? Salah satu alasannya adalah karena perubahan satu byte saja pada pesan awal dapat mengakibatkan perubahan besar pada tanda tangan digital yang akan digunakan. Selain itu juga terdapat fakta bahwa XML sejauh ini belum memiliki standar yang benar-benar mengikat karena masih dalam tahap dieksplorasi. Jadi, tag <baru sekali> dan < baru_sekali > bisa dianggap sama, bisa dianggap tidak sama.

Untuk mengatasi hal ini, satu-satunya cara yang mungkin dapat dilakukan adalah memberikan standardisasi XML. Dengan misalnya memberikan aturan bahwa sebuah tag XML

tidak boleh memiliki karakter spasi di dalamnya, atau standard-standard baru lainnya akan sangat membantu proses penandatanganan dokumen XML tersebut.

Usaha lain yang mungkin dapat disarankan adalah menemukan algoritma yang lebih mangkus untuk melakukan *canonicalization* pada XML supaya tidak terlalu mengganggu proses transaksi. Namun, upaya ini mungkin tidak semudah upaya yang pertama.

Selama proses *canonicalizaion* XML merupakan sedikit beban, penandatanganan dokumen-dokumen berbasis XML mungkin akan sedikit terhambat, namun berhubung pemakaian struktur XML akan semakin berkembang untuk tahun-tahun mendatang, mudah-mudahan hal ini akan segera teratasi.

**5. Kesimpulan**

- XML DSIG memiliki elemen-elemen yang dapat membantu melakukan otentifikasi pengiriman data-data XML.
- *SignedInfo* mengandung atau mereferensikan data yang ditandatangani dan menspesifikasikan algoritma apa yang digunakan.
- *CanonicalizationMethod* dan *SignatureMethod* digunakan oleh elemen *SignatureValue* dan disertakan dalam *SignedInfo* untuk melindungi mereka dari gangguan.
- Satu atau lebih elemen *Reference* menentukan sumber daya yang ditandatangani dan perubahan apapun yang akan diterapkan pada pesan awal sebelum penandatanganan.
- *DigestMethod* menentukan algoritma hash sebelum menerapkan hash. *DigestValue* berisi hasil penerapan algoritma hash ke pesan awal.
- *SignatureValue* adalah elemen yang berisi hasil *encode* tanda tangan dalam Base64
- *KeyInfo* adalah elemen yang secara opsional memungkinkan pengirim untuk memberikan penerima kunci yang memvalidasi tanda tangan, biasanya dalam bentuk satu atau lebih sertifikat digital X.509.
- Elemen *Object* (opsional) berisi data yang ditandatangani jika tanda tangan ini bersifat *enveloping* (membungkus) data awal.
- *Canonicalization* (disingkat c14n, dimana 14 merupakan jumlah huruf antara C dan N) adalah suatu proses untuk mengkonversi data yang memiliki lebih dari satu perwakilan menjadi "standar" kanonik perwakilan.
- Selama proses *canonicalizaion* XML merupakan sedikit beban, penandatanganan dokumen-dokumen berbasis XML mungkin akan sedikit terhambat, namun berhubung pemakaian struktur XML akan semakin berkembang untuk tahun-tahun mendatang, mudah-mudahan hal ini akan segera teratasi.

**DAFTAR PUSTAKA**


[1] Understanding XML Digital Signature http://msdn.microsoft.com/en-us/library/ms996502.aspx Tanggal Akses: 6 Mei 2009 21.00.

[2] Munir, Rinaldi. (2006). Diktat Kuliah IF5054 Kriptografi. Program Studi Teknik Informatika, Institut Teknologi Bandung.

[3] Srivastava, Manoj. (2005). Digital Signature in Web Application. www.infomosaic.net/Downloads/Srivastava_webappdigitalsignature_nov_08_2005.pdf Tanggal Akses: 5 Mei 2009 19.00.