XML Consortium

～ 第5回 XMLコンソーシアムWeek ～

sPlatプロジェクト
暗号化XMLデータ利用技術についての課題と対策
実装編

2006年5月24日

XMLコンソーシアム Webサービス実証部会
荒本 道隆（アドソル日進株式会社）

実装編のゴール

- WS-SecurityでのEnd-to-Endセキュリティの実現
  - 実装しようとすると、色々な課題が出てきた
  - 汎用的な解決策・実装を作成する
- XSLTによるスキーマ変換ツールを実装
  - 今回はスキーマ変換方式を選択
    - スキーマ変換方式、ハンドラー方式、エンジン実装方式
  - 元スキーマ(xsd)をXSLTで変換する
  - 変換パターンの洗い出し
  - 実際に実装してみて、制限事項や問題点を確認
- 妥当性検証とデータバインディングの両方を検討
  - 妥当性検証：**MUST**と**MAY**
  - データバインディング：Javaのclassにマッピングした時など
- 「どこを暗号化する」の決定は対象外
  - 暗号化対象（**MUST**,**MAY**）の自動決定は不可能
  - ビジネス、セキュリティの両観点から要件を決定する必要がある
  - 利用シーンによって、セキュリティ要件が異なる

XML Consortium

XSLTによるスキーマ変換を実装

- XSLTによるスキーマ変換ツール
  - 元スキーマ（xsd）に対してXSLTにより変換し、XML暗号に対応した変換後スキーマ（xsd）を作成する
  - 変換箇所は、元スキーマに対するXPathで指定
    - <xsl:template match="xs:element[@name='CreditCardInformation']
      /xs:complexType/xs:sequence/xs:element[@ref='CreditCardNumber']">
  - インスタンス（データXML）ではなく、元スキーマに対するポリシーを記述
    - スキーマを理解しないと、ポリシーを記述できない
    - インスタンスに対するポリシーを記述するには、別方式の検討が必要
  - **変換後の名前空間は、同じものを使用する**
    - 名前空間を変えると、署名が影響を受けてしまう
    - バインドした場合、異なるpackageになるのでコピーできない
  - 変換パターンを準備して、そこから選択
    - ノウハウとして活用できる

スキーマ変換方式によるメリット

- Webサービス実装のほとんどは、スキーマを参照する
  - 変換後スキーマを準備すれば、多くの実装に対応可能
    - 他の方式では、Webサービスの実装に依存してしまう
    - 選択した方式に対応していない実装は、対象外に
  - Webサービス以外にもスキーマを利用する物に利用できる
    - XML-DBなど
- XSLTを利用できる環境が豊富
  - XSLTを別途実行しても問題ない
- XSLT,XPathを使用
  - DOMを直接操作するよりは、分かり易い
- 自動化によるメリット
  - 元スキーマに変更があった場合に、反映が楽
  - 通信相手ごとにセキュリティポリシーが違う場合
  - 数箇所程度であれば、手動で編集した方が早いし確実ではあるが…
    - 変換後から、ルールを記述する手間
    - 予想外の部分も変換される危険性（記述ミス）

スキーマ変換方式によるデメリット

- 名前空間の問題
  - 変換後も同じ名前空間を使うので、元スキーマと変換後スキーマを同一アプリ内で使うと、バッティングしてしまう
  - スキーマから自動生成されるclassのパッケージ名を意識してみる
- 暗号化される要素を静的に指定
  - 変更する時には再デプロイが必要
  - 動的に変化する場合は、可能性のある要素すべてをMAYとして定義する

暗号化例（インスタンス）

- CreditCardInformationを暗号化する場合
  - CreditCardNumber は暗号化必須（MUST）
  - ExpireDateは暗号化任意(MAY1)
  - CardHolderNameは暗号化任意(MAY2)

暗号化前

```
<CreditCardInformation>
   <CreditCardAuthority>XYZ</CreditCardAuthority>
   <CreditCardNumber>0123456789</CreditCardNumber>
   <ExpireDate>2008-12</ExpireDate>
   <CardHolderName>Aramoto Michitaka</CardHolderName>
</CreditCardInformation>
```

MUST / MAY1 / MAY2

暗号化

暗号化後

```
<CreditCardInformation>
   <CreditCardAuthority>XYZ</CreditCardAuthority>
   <xenc:EncryptedData Type="http://..."
      <xenc:EncryptionMethod Algorithm="http://..."
      <xenc:CipherData>
         <xenc:CipherValue>fhRzmys1...</xenc:CipherValue>
      </xenc:CipherData>
   </xenc:EncryptedData>
   <ExpireDate>2008-12</ExpireDate>
   <CardHolderName>Aramoto Michitaka</CardHolderName>
</BookingInfo>
```

暗号化例（スキーマ）

- 元スキーマ

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
          targetNamespace="http://www.xmlconsortium.org/bukai/ouyou/demo/travel"
          xmlns="http://www.xmlconsortium.org/bukai/ouyou/demo/travel"
          elementFormDefault="unqualified">

 <xs:element name="CreditCardInformation">
  <xs:complexType>
   <xs:sequence>
    <xs:element ref="CreditCardAuthority" minOccurs="0"/>
    <xs:element ref="CreditCardNumber" minOccurs="0" />
    <xs:element ref="ExpireDate" minOccurs="0" />
    <xs:element ref="CardHolderName" minOccurs="0" />
   </xs:sequence>
  </xs:complexType>
 </xs:element>

 <xs:element name="CreditCardAuthority" type="xs:string"/>
 <xs:element name="CreditCardNumber" type="xs:string" />
 <xs:element name="ExpireDate" type="xs:gYearMonth"/>
 <xs:element name="CardHolderName" type="xs:string"/>
</xs:schema>
```

MUST

MAY1

MAY2

他で使っていないので、隠す

XML Consortium

暗号化例（スキーマ）

- 変換後スキーマ

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://www.xmlconsortium.org/bukai/ouyou/demo/travel"
targetNamespace="http://www.xmlconsortium.org/bukai/ouyou/demo/travel" elementFormDefault="unqualified">
<xsl:import xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
namespace="http://www.w3.org/2001/04/xmlenc#" schemaLocation="xenc-schema.xsd" />

 <xs:element name="CreditCardInformation">
  <xs:complexType>
   <xs:sequence>
    <xs:element ref="CreditCardAuthority" minOccurs="0"/>
    <xs:element xmlns:xenc="http://www.w3.org/2001/04/xmlenc#" ref="xenc:EncryptedData" minOccurs="0" />
     <xs:choice minOccurs="0">
      <xs:element ref="ExpireDate" />
      <xs:element xmlns:xenc="http://www.w3.org/2001/04/xmlenc#" ref="xenc:EncryptedData" />
     </xs:choice>
     <xs:choice>
      <xs:element ref="CardHolderName" minOccurs="0" />
      <xs:element xmlns:xenc="http://www.w3.org/2001/04/xmlenc#" ref="xenc:EncryptedData" minOccurs="0" />
     </xs:choice>
   </xs:sequence>
  </xs:complexType>
 </xs:element>

 <xs:element name="CreditCardAuthority" type="xs:string"/>

 <xs:element name="ExpireDate" type="xs:gYearMonth"/>
 <xs:element name="CardHolderName" type="xs:string"/>
</xs:schema>
```

xencのスキーマ定義

MUST

MAY2

MAY1

他で使っていないので、隠す

暗号化例 (XSLT) (1)

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:xenc="http://www.w3.org/2001/04/xmlenc#" version="1.0">

<xsl:output method="xml" encoding="UTF-8" indent="yes" />

<xsl:template match="/">
   <xsl:apply-templates/>
</xsl:template>

<xsl:template match="/xs:schema">
   <xsl:copy>
      <!-- まずschemaの属性をすべてコピーする -->
      <xsl:apply-templates select="@*"/>

      <!-- xencのスキーマのインポートを記述 -->
      <xsl:element name="xsl:import">
         <xsl:attribute name="namespace">http://www.w3.org/2001/04/xmlenc#</xsl:attribute>
<!--        <xsl:attribute name="schemaLocation">http://www.w3.org/TR/2002/REC-xmlenc-core-
20021210/xenc-schema.xsd</xsl:attribute> -->
         <xsl:attribute name="schemaLocation">xenc-schema.xsd</xsl:attribute>
      </xsl:element>

      <!-- 子要素をコピーする -->
      <xsl:apply-templates select="node()"/>
   </xsl:copy>
</xsl:template>
```

xencのスキーマ定義

暗号化例 (XSLT) (2)

- 変換箇所の指定

MUST

```
<!-- エレメント暗号、暗号化必須項目 -->
<xsl:template
match="xs:element[@name='CreditCardInformation']/xs:complexType/xs:sequence/xs:element[@ref='CreditC
ardNumber']">
   <xsl:call-template name="xenc-element-must" />
</xsl:template>

<!-- CreditCardNumberのelement定義を削除する -->
<xsl:template match="xs:element[@name='CreditCardNumber']" />

<!-- エレメント暗号、暗号化してもしなくても良い項目、その1 -->
<xsl:template
match="xs:element[@name='CreditCardInformation']/xs:complexType/xs:sequence/xs:element[@ref='ExpireD
ate']">
   <xsl:call-template name="xenc-element-may1" />
</xsl:template>

<!-- エレメント暗号、暗号化してもしなくても良い項目、その2 -->
<xsl:template
match="xs:element[@name='CreditCardInformation']/xs:complexType/xs:sequence/xs:element[@ref='CardHol
derName']">
   <xsl:call-template name="xenc-element-may2" />
</xsl:template>
```

他で使っていないので、隠す

MAY2

MAY1

暗号化例 (XSLT) (3)

- MUST変換

```
<!-- エレメント暗号、暗号化必須項目
   <xs:element ref="XXXXXXXXXX" minOccurs="m" maxOccurs="n" />
   ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓
   <xs:element ref="xenc:EncryptedData" minOccurs="m" maxOccurs="n" />
-->
<xsl:template name="xenc-element-must">
   <xs:element ref="xenc:EncryptedData">
      <xsl:if test="@minOccurs!="">
         <xsl:attribute name="minOccurs">
            <xsl:value-of select="@minOccurs" />
         </xsl:attribute>
      </xsl:if>
      <xsl:if test="@maxOccurs!="">
         <xsl:attribute name="maxOccurs">
            <xsl:value-of select="@maxOccurs" />
         </xsl:attribute>
      </xsl:if>
   </xs:element>
</xsl:template>
```


暗号化例 (XSLT) (4)

- MAY1変換

```
<!-- エレメント暗号、暗号化してもしなくても良い項目、その1
   <xs:element ref="XXXXXXXXXX" minOccurs="m" maxOccurs="n" />
   ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓
   <xs:choice minOccurs="m" maxOccurs="n">
      <xs:element ref="XXXXXXXXXX" />
      <xs:element ref="xenc:EncryptedData" />
   </xs:choice>
-->
<xsl:template name="xenc-element-may1">
   <xsl:element name="xs:choice">
      <xsl:if test="@minOccurs!="">
         <xsl:attribute name="minOccurs">
            <xsl:value-of select="@minOccurs" />
         </xsl:attribute>
      </xsl:if>
      <xsl:if test="@maxOccurs!="">
         <xsl:attribute name="maxOccurs">
            <xsl:value-of select="@maxOccurs" />
         </xsl:attribute>
      </xsl:if>

      <xsl:element name="xs:element">
         <xsl:attribute name="ref"><xsl:value-of select="@ref" /></xsl:attribute>
         <!-- 該当する要素の属性と子要素をそのままコピーする -->
         <xsl:apply-templates select="text()|child::node()"/>
      </xsl:element>
      <xs:element ref="xenc:EncryptedData" />
   </xsl:element>
</xsl:template>
```

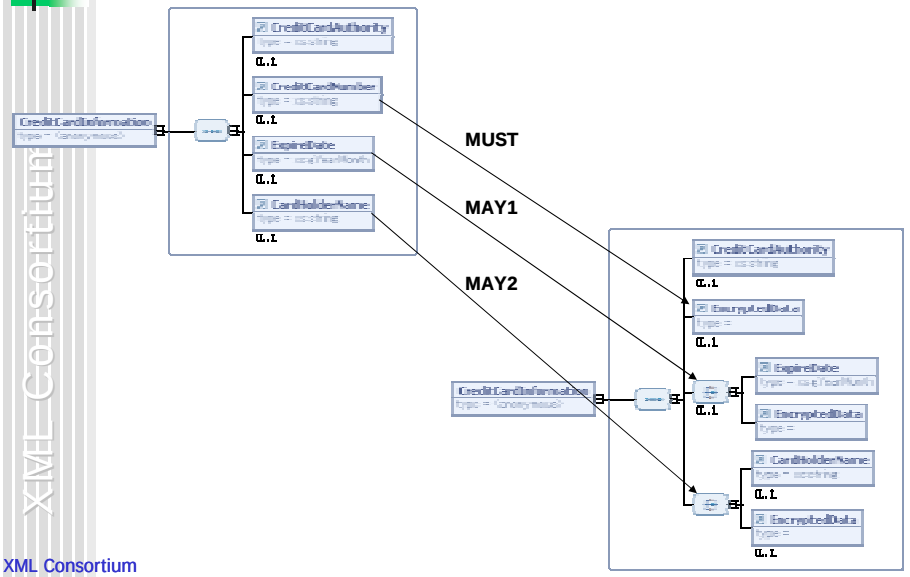
暗号化例 (XSLT) (5)

- MAY2変換

```
<!-- エレメント暗号、暗号化してもしなくても良い項目、その2
   <xs:element ref="XXXXXXXXXX" minOccurs="m" maxOccurs="n"/>
   ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓
   <xs:choice>
      <xs:element ref="XXXXXXXXXX" minOccurs="m" maxOccurs="n"/>
      <xs:element ref="xenc:EncryptedData" minOccurs="m" maxOccurs="n"/>
   </xs:choice>
-->
<xsl:template name="xenc-element-may2">
   <xsl:element name="xs:choice">
      <xsl:element name="xs:element">
         <xsl:attribute name="ref"><xsl:value-of select="@ref" /></xsl:attribute>
         <!-- 該当する要素以下をそのままコピーする -->
         <xsl:apply-templates select="@*|node()"/>
      </xsl:element>
      <xs:element ref="xenc:EncryptedData">
         <xsl:if test="@minOccurs!=''">
            <xsl:attribute name="minOccurs">
               <xsl:value-of select="@minOccurs" />
            </xsl:attribute>
         </xsl:if>
         <xsl:if test="@maxOccurs!=''">
            <xsl:attribute name="maxOccurs">
               <xsl:value-of select="@maxOccurs" />
            </xsl:attribute>
         </xsl:if>
      </xs:element>
   </xsl:element>
</xsl:template>

</xsl:stylesheet>
```


変換結果例 (スキーマ)



demo

- WS-Security妥当性検証デモ
  - スキーマ：TravelXML（1.1.1）
  - 環境：WTP 1.0.2（Eclipse 3.1.2）+Tomcat-Plugin
  - サーバ：Tomcat5.5.17+Axis1.3
  - クライアント：telnetコマンド
  - 1．通常のSOAPメッセージが届くことを確認
  - 2．暗号化済みのメッセージでエラーが出ることを確認
  - 3．スキーマを変換（MUST&MAY）し、それを適用
  - 4．暗号化済みのメッセージでエラーが出ないことを確認
  - 5．通常のメッセージでエラーが出ることを確認（MUSTの部分）

注意事項

- 元スキーマを知られたくない場合もある
  - スキーマを一般公開していない
  - 「この暗号化されたデータの中に、クレジットカード番号がある」と分かった場合の危険度
    - エレメント暗号（タグごと暗号化）した意味がない
  - 変換後のスキーマだけを配布
    - ポリシー伝達方式に発展させる時にも、注意が必要
- 変換前と変換後のスキーマを同一環境で使用する場合
  - 同じ名前空間なので、同居が困難
  - インスタンス（データ）だけを見ても、どちらのスキーマに準拠したデータなのか、判断できない
    - インスタンスの種類は、名前空間を見れば分かる
    - 異なるポリシーによるインスタンス（データ）を受け付けるには、エンドポイントを別にする

今回の実装の制限事項(1)

- すべてのパターンを網羅できていない
  - スキーマの記述方法は、作成者によって異なる
  - 同じインスタンスを表すためのスキーマでも、違う書き方ができる
  - TravelXML1.1.1を対象として、変換パターンを作成
- 変換結果の確認方法
  - axis1.3によるwsdl2javaの実行結果で確認

今回の実装の制限事項(2)

- SOAPヘッダはどうするのか？
  - 今回は対象外とする

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
 <Header:Header xmlns:Header="http://schemas.xmlsoap.org/soap/envelope/">
  <wsse:Security Header:mustUnderstand="1" xmlns:wsse="http://schemas.xmlsoap.org/ws/2002/07/secext">
   <xenc:EncryptedKey xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
    <EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-1_5"
xmlns="http://www.w3.org/2001/04/xmlenc#" xmlns:ds="http://www.w3.org/2000/09/xmldsig#"/>
    <KeyInfo xmlns:dsig="http://www.w3.org/2000/09/xmldsig#">
     <wsse:SecurityTokenReference>
      <wsse:KeyIdentifier>.....</wsse:KeyIdentifier>
     </wsse:SecurityTokenReference>
    </KeyInfo>
    <CipherData xmlns="http://www.w3.org/2001/04/xmlenc#"
xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
     <CipherValue>.....</CipherValue>
    </CipherData>
    <ReferenceList>
     <DataReference URI="#....."/>
    </ReferenceList>
   </xenc:EncryptedKey>
  </wsse:Security>
 </Header:Header>
 <soapenv:Body>
  <AllotmentBookingReport xmlns="http://www.xmlconsortium.org/bukai/ouyou/demo/travel">
   .....
  </AllotmentBookingReport>
 </soapenv:Body>
</soapenv:Envelope>
```

Webサービスの実装によって操作方法が異なるので、今回は対象外

今回の実装の制限事項(3)

- 複数のスキーマから、1つのスキーマを参照した場合
  - wsdlに、複数の<xs:schema>を記述
  - 複数のスキーマファイルが、1つのスキーマファイルを参照
  - スキーマによって、参照しているスキーマの暗号化対象を変えたい場合には、対応できない

```
<xs:schema
xmlns:xs="http://www.w3.org/2001/XMLSchema" targetNamespace="http://sPlat.xmlconsortium.org/OrderSheet" xmlns:tns="http://sPlat.xmlconsortium.org/OrderSheet">

  <xs:complexType name="OrderSheetType">
    <xs:sequence>
      <xs:element name="CreditCard" type="tns:CreditCardType" minOccurs="0" />
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="CreditCardType">
    <xs:sequence>
      <xs:element name="CardType" type="xs:string" />
      <xs:element name="CardNumber" type="xs:string" />
      <xs:element name="ExpireDate" type="xs:string" />
      <xs:element name="HolderName" type="xs:string" />
    </xs:sequence>
  </xs:complexType>
</xs:schema>


<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" targetNamespace="http://sPlat.xmlconsortium.org/OrderSheet"
xmlns:tns="http://sPlat.xmlconsortium.org/OrderSheet">

  <xs:element name="OrderSheet1" type="c:OrderSheetType" />
  <xs:element name="OrderSheet2" type="c:OrderSheetType" />

</xs:schema>
```

妥当性検証は何とかなっても、バインドはどうする？

こっち側だけCardNumberを暗号化必須にしたい

苦労した点

- XSLTで、新しい名前空間を追加する書き方
  - 「xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"」を追記する必要がある
  - XSLTで既存要素に「xmlns:xxx="～"」を付加する事はできない

> XSLT プロセッサは、生成したアトリビュートを XML として出力するためのプレフィックスを選択する際に、name アトリビュートに指定した QName のプレフィックスを使用してもよい。ただし、これは必須ではなく、プレフィックスが xmlns の場合には使用してはならない (must)。以下の例は、実行してもエラーではないが、ネームスペース宣言は出力されない。
>
> <xsl:attribute name="xmlns:xsl" namespace="whatever">http://www.w3.org/1999/XSL/Transform</xsl:attribute>

http://www.infoteria.com/jp/contents/xml-data/REC-xslt-19991116-jpn.htm
XSL Transformations (XSLT)
バージョン 1.0
W3C 勧告 1999 年 11月 16 日
より

- 各「xenc:EncryptedData」要素に、「xmlns:xenc="～"」を記述
  - スキーマの意味としては、同一
  - 詳しくは、変換後スキーマを参照

実装編のまとめ

- スキーマ変換方式で実現できる範囲
  - 静的なセキュリティポリシーの適用
  - Webサービス以外への転用
- 別書式から、XSLを生成するのは難しくない
  - 変換方式（MUST,MAY）とXPathを指定
  - 設定ファイル化→ポリシー方式への発展
- インスタンス（データ）に対するXPath指定は、スキーマ変換方式では難しい
  - スキーマを解析するエンジンを実装しなければならない
    - 限られたリソースでは、1から実装するのは無理
    - オープンソースの物に機能追加できないか
  - 他方式（ハンドラー、エンジン）であれば、実現できそう
- 結局、いつも名前空間で苦労する….
  - 「元スキーマと変換後スキーマの名前空間を同じにするか？」について、最も多くの時間をかけて議論した
  - 過去の実証実験でも、いつも名前空間で苦労した

添付資料（XSLTの実行方法）

- Java1.5+Xalan2.7.0の場合

```
set CLASSPATH=xalan.jar;%CLASSPATH%
set CLASSPATH=xercesImpl.jar;%CLASSPATH%

java org.apache.xalan.xslt.Process -in 元xsdファイル.xsd -out 出力xsdファイル.xsd -xsl xsltファイル.xsl
```