社団法人 情報処理学会 研究報告
IPSJ SIG Technical Reports

2006－ＤＤ－56（７）
2006／7／28

XSLTを用いた暗号化XMLデータのスキーマ検証方式の実装

荒本 道隆† 中山 弘二郎†† 大場 みち子†††

†アドソル日進（株） 生産技術部 〒108-0075 東京都港区港南4－1－8 リバージュ品川
††（株）日立製作所 システム開発研究所 〒215-0013 神奈川県川崎市麻生区王禅寺1099
†††（株）日立製作所 ソフトウェア事業部 〒140-8573 東京都品川区南大井6-26-2 大森ベルポートA館
E-mail: †Aramoto.Michitaka@adniss.jp, ††kojiro@sdl.hitachi.co.jp, †††michiko.oba.cq@hitachi.com

あらまし XML暗号を用いて部分暗号化されたXMLデータは，スキーマに対する妥当性が失われているため，スキーマ検証が実施できず，スキーマを元にしたバインディングも行えない．end-to-endのセキュリティを実現するためには，中継者はXMLデータそのものを直接操作する必要があった．本研究では，XSLTを用いて暗号化に対応したスキーマを自動生成することで，部分暗号化されたXMLデータのスキーマ検証とバインディングを行う方法を提案する．

キーワード XML，Webサービス，セキュリティ，XML暗号，スキーマ検証，XSLT

Implementation of Schema Validation Mechanism Using XSLT for Encrypted XML Data

Michitaka Aramoto† Kojiro Nakayama†† and Michiko Oba†††

† Productive Engineering Dept., Ad-Sol Nissin Corp. 4-1-8 Kounan Riverge, Shinagawa, Minato-ku, Tokyo 108-0075 Japan
†† Systems Development Laboratory, Hitachi, Ltd. 1099 Ohzenji, Asao-ku, Kawasaki-shi, Kanagawa 215-0013 Japan
††† Software Division, Hitachi, Ltd. Omori Bellport A Bldg. 6-26-2 Minamioi, Shinagawa-ku, Tokyo 140-8573 Japan
E-mail: †Aramoto.Michitaka@adniss.jp, ††kojiro@sdl.hitachi.co.jp, †††michiko.oba.cq@hitachi.com

Abstract The XML data from which the part is encrypted cannot perform of the schema verification because validity to the schema is lost, and cannot do binding based on the schema. To achieve the end-to-end security, those who relaid it had to operate the XML data directly. In this paper, it proposes the method of doing the schema verification and binding of the XML data from which the part is encrypted by generating the schema corresponding to the encryption automatically by the use of XSLT.

Keyword XML, Web Services, Security, XML Encryption, Schema Validation, XSLT

1. はじめに

Webサービスの仕様が発表されてから6年が経過し，その後もWebサービスには様々な関連仕様が追加され続けている．Webサービスは当初，それまでのB2B（企業間取り引き）で使われてきたEDI（Electronic Data Interchange）やCSV（Comma Separated Values）などと比較すると，より簡単に他の企業と接続ができると言われてきた．さらに，UDDIによる動的な連携先検索や，アグリゲーション等による新しいビジネスモデルも示唆されてきた．実際，企業内の異なる部署間や異なるシステム間，またグループ企業内でのシステム間連携の手段として多くのWebサービスが利用されてきている．しかし，残念ながら企業間取り引きにあまりWebサービスは使われていない．

XMLコンソーシアムでは，Webサービスを使った実証実験を2001年以降7回に渡り実施してきており，様々なWebサービスの実装間での相互接続を行ってきた．一部の実装の間では，WSDL（Web Services Description Language）からスケルトンやスタブを自動生成した後にコードを追記する必要があったりもしたが，最終的に「接続できない」ということはなく，技術的にはWebサービスにおける相互接続性は十分に確保されていると言える．

WebサービスをB2BやB2C（企業と一般消費者の取り引き）で安心して使うためには，セキュリティの確保が必須である．既存のセキュリティ技術を使い，httpsによりSSL（Secure Socket Layer）あるいはTLS（Transport Layer Security）により通信内容の暗号化と改竄防止を行うこともできるが，それでは

Point-to-Point のセキュリティしか確保できない．Point-to-Point のセキュリティとは，通信経路の途中での盗聴や改竄はほぼ不可能であるが，中継者はすべてのデータを参照・変更が出来てしまう．複数の企業間に渡る商取引を安全に行うためには，End-to-End のセキュリティを確保し，中継者に対しても暗号化と改竄防止を行えなければならない．

Web サービスの関連仕様では，WS-Security によりメッセージの完全性と秘匿性が確保され，End-to-End のメッセージレベルでのセキュリティが実現できる．完全性とは，XML-Signature の署名により作成者の身元を証明し，XML ドキュメントが改竄されていない事を保証する．秘匿性とは，XML-Encryption の暗号化により第三者により盗み読みされない事を保証する．End-to-End のセキュリティを実現しようとした場合には，送信者や受信者は自身が必要とするセキュリティ機能だけを実装すれば良いのだが，中継者は自身が使用しないセキュリティにも対応しなければメッセージの送受信ができない．送信者と受信者は XML ドキュメントに関する仕様である XML-Signature や XML-Encryption のための既存のライブラリが流用できるが，中継者は標準的な実装方法が存在していないために End-to-End のセキュリティを実現しようとするときに，さまざまな課題に直面する．

本稿では，中継者において End-to-End のセキュリティを実現するための，既存の Web サービスの実装での解決方法を述べる．

2. スキーマ用言語

XML の構造を定義するスキーマ用言語には，DTD（Document Type Definition），XSD（XML Schema Definitions），RELAX（REgular Language description for XML），XML Data Reduced など，さまざまな言語があるが，Web サービスのインターフェイスを定義する WSDL では「XML Schema Definitions を規範的な型システムとして使用することが望ましい」としているので，本稿では XSD のみを対象とする．

3. 中継者に求められる機能

図１に送信者，中継者，受信者の関係を示す．送信者は送信しようとするインスタンスに対し，１）XML ドキュメント化，２）妥当性検証，３）暗号化，を行ってから送信する．また，受信者は受信したメッセージに対し，１）復号化，２）妥当性検証，３）インスタンスにマッピングする事で，送信者のインスタンスを取得できる．この時，送信者と受信者の妥当性検証は暗号化されていない状態の XML ドキュメントに対して行えるので，問題なく検証が成功する．対して中継者は，秘匿性のために暗号化も復号化も行えず，暗号化された状態のメッセージに対して妥当性検証を行わなくてはならない．また「秘匿性が保たれているか？」を保証するために，暗号化されているべき箇所が確実に暗号化されているかどうかの検証も行わなくてはならない．そのため，XML ドキュメントが暗号化されているかを確認し，それを受信者に中継する機能が必要となる．

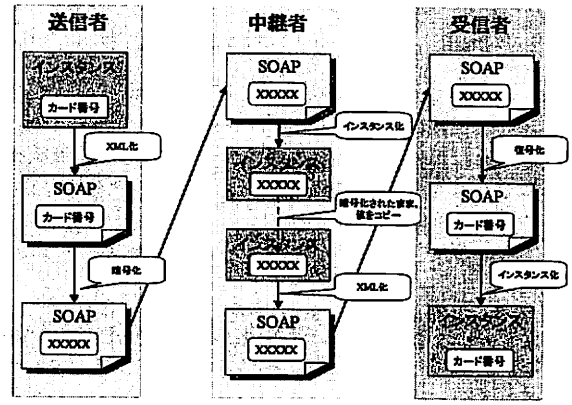

図１　end-to-end の秘匿性
Fig.1　end-to-end secretly

図２の SOAP メッセージを WS-Security により暗号化すると，図３のように暗号化した箇所の構造が変化してしまう．

```
<?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/">
  </Header:Header>
  <soapenv:Body>
    <AllotmentBookingReport xmlns="http://www.xmlconsortium.org/bukai/ouyou/demo/travel">
      --省略--
      <BasicRateInformation>
        <TaxServiceFee xsi:nil="true"/>
        <CreditCardInformation>
          <CreditCardAuthority>XYZ</CreditCardAuthority>
          <CreditCardNumber>0123456789</CreditCardNumber>
          <ExpireDate>2008-12</ExpireDate>
          <CardHolderName>Aramoto Michitaka</CardHolderName>
        </CreditCardInformation>
      </BasicRateInformation>
      --省略--
    </AllotmentBookingReport>
  </soapenv:Body>
</soapenv:Envelope>
```

図２　元 SOAP メッセージ
Fig.2　Original SOAP message

```
<?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">
      --省略--
    </wsse:Security>
  </Header:Header>
  <soapenv:Body>
    <AllotmentBookingReport xmlns="http://www.xmlconsortium.org/bukai/ouyou/demo/travel">
      --省略--
      <BasicRateInformation>
        <TaxServiceFee xsi:nil="true"/>
        <CreditCardInformation>
          <CreditCardAuthority>XYZ</CreditCardAuthority>
          <xenc:EncryptedData Id="G4154298D" Type="http://www.w3.org/2001/04/xmlenc#Element" xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
            <EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#tripledes-cbc" xmlns="http://www.w3.org/2001/04/xmlenc#" xmlns:ds="http://www.w3.org/2000/09/xmldsig#"/>
            <CipherData xmlns="http://www.w3.org/2001/04/xmlenc#" xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
              <CipherValue>NEY3TOLOxV89Tw7n3OYUOyW.....</CipherValue>
            </CipherData>
          </xenc:EncryptedData>
          <ExpireDate>2008-12</ExpireDate>
          <CardHolderName>Aramoto Michitaka</CardHolderName>
        </CreditCardInformation>
      </BasicRateInformation>
      --省略--
    </AllotmentBookingReport>
  </soapenv:Body>
</soapenv:Envelope>
```

図3　暗号化済み SOAP メッセージ
Fig.3　Encrypted SOAP message

XML ドキュメントに対する部分暗号化の方式には2種類ある．1つはエレメント暗号で，暗号化対象の1つの要素とその下位要素を暗号化し，元要素のタグごと<xenc:EncryptedData>に置換する．もう1つはコンテント暗号で，暗号化対象の1つの要素の内容とその下位要素を暗号化し，元要素のタグを残して<xenc:EncryptedData>に置換する．図3はエレメント暗号を使用した例である．このように，CreditCardNumber に代わり，元スキーマには存在しない<xenc:EncryptedData>というタグが出てくるため，暗号化された XML ドキュメントに対して元スキーマを使用して妥当性検証を行うと，検証に失敗する．中継者は<xenc:EncryptedData>も検証を成功させ，そのままの形で中継できなければ，受信者が正しく受信できない．

Web サービスのリクエスタ（送信者）やプロバイダ（受信者）は，WSDL+XSD を読み込んで送信用スタブや受信用スケルトンを自動生成する．このスキーマファイルに<xenc:EncryptedData>に関する記述を行うことで，それを元に自動生成した送信用スタブや受信用スケルトンは，暗号化された XML ドキュメントを扱えるようになる．

4. XSLT によるスキーマの変換

中継者が暗号化された XML ドキュメントを扱うには，元スキーマ（XSD）ファイルに対して XSLT 変換を行い，XML 暗号に対応した変換後スキーマ（ポスト暗号化スキーマ）を作成する事で実現する．変換後スキーマには<xenc:EncryptedData>に関する記述を追加しているので，暗号化された XML ドキュメントを扱えるようになる．

元スキーマに対し，どこを暗号化してどのように変換するかの指定が，セキュリティのポリシーとなる．中継者の中継先が複数ある場合，受信者ごとにポリシーが異なる可能性がある事に注意しなければならない．異なるポリシーを適用した変換後スキーマがすべて同じ名前空間を持ってしまうため，どのポリシーに適用した XML ドキュメントかを判別することができない．そのため，ポリシーごとに Web サービスのエンドポイントをそれぞれ別にするなどの対応が必要である．

どこを暗号化する必要があるか，どのようにしてポリシーを伝達するかは，ビジネス・セキュリティの両観点によって決定されるので，本稿では対象外とする．

スキーマファイルの変換は，事前に静的に行うことを前提としている．ポリシーの変更時には，中継者は Web サービスの再デプロイが必要である．

5. XSLT を使う利点

XSLT（XSL Transformation）は XML ドキュメントの構造を変換するための言語である．XSLT エンジンにより，XML ドキュメントを XML 以外にも，HTML, CSV, テキストなど他の形態に変換する事ができる．

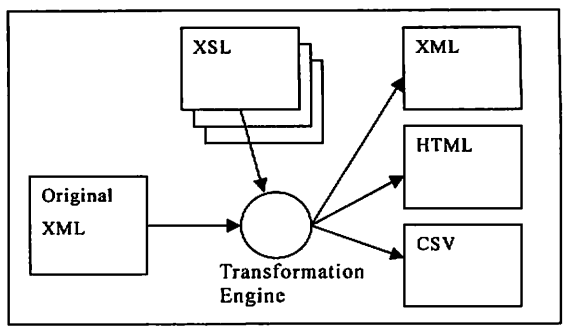

図4　XSLT 処理の概要
Fig.4　Outline of XSLT processing

XSLT は，XML から HTML への変換などに広く利用されており，XML を処理する環境のほとんどで XSLT エンジンが実行可能である．XSLT は標準化されているので，特定の環境や言語に依存する心配もない．

6. XML の名前空間

XML では独自のタグセットを自由に設計できるため，同じタグ名が違う意味で使われる可能性がある．そこで，それぞれのタグセットに固有の URI を割り当

て，タグ名を URI で修飾することで区別する．XML 名前空間の URI が違えば，同じタグ名であっても違うものとなる．

本方式では，元スキーマを XSLT によって変換した後も，名前空間は変更しない．もし，変換後のスキーマに異なる名前空間を割り当てれば，名前空間を見ただけで「部分暗号化された XML ドキュメント」と識別する事ができる．しかし，全体の名前空間を変更すると，部分暗号化とは関係ない部分の名前空間も変わってしまい，元名前空間を持つデータとは同等では無くなってしまう．例えば Java であれば，名前空間とはバインディングした時の Package 名にあたるので，名前空間 old は Package old に，名前空間 new は Package new に，それぞれバインドされる．そこに a というインスタンスがある場合，old.a と new.a はパッケージ名が違うので，まったく違うものとして扱われることになる．

変換後スキーマは，元スキーマの名前空間に加え，<xenc:EncryptedData>とその下位要素が定義されている"http://www.w3.org/2001/04/xmlenc#"の名前空間を持つ．

7. 変換後スキーマの再利用

本方式では，XSLT により暗号化に対応した変換後スキーマを自動生成する．中継者は暗号化されている XML ドキュメントを復号化できないため，全ての処理において，この変換後スキーマに記述されている構造で扱わなければならない．例えば，XML-DB に受信した XML ドキュメントを格納するには，変換後スキーマの構造を使うことで効率良く正確に格納できる．一部の XML-DB ではスキーマ構造を必要としない物もあるが，データ格納時に妥当性検証が行えないなどの制限が発生してしまうため，スキーマ構造を利用するメリットは大きい．

8. 変換パターン

同一の XML ドキュメント構造に対しても異なる記述方法でスキーマ定義を行うことができる．図５で示したサンプル XML データに対するスキーマを記述するにも，大別するとローカル宣言を使う図６「Russian-Doll」と，グローバル宣言を使う図７「Salami-Slice」の２通りの方法で記述できる．どちらを使っても，同じ XML データの構造をそれぞれ定義できる．さらに，これらの２つを混合して記述したり，別ファイルを include や import するなど，様々な記述方法が可能である．

```
<?xml version="1.0" encoding="UTF-8"?>
<CreditCardInformation xmlns="http://www.xmlconsortium.org/bukai/ouyou/demo/PackTour"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.xmlconsortium.org/bukai/ouyou/demo/PackTour">
    <CreditCardAuthority>XYZ</CreditCardAuthority>
    <CreditCardNumber>0123456789</CreditCardNumber>
    <ExpireDate>2008-12</ExpireDate>
    <CardHolderName>Aramoto Michitaka</CardHolderName>
</CreditCardInformation>
```

図５　XML データ
Fig.5　XML data

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

図６　ロシアンドール スキーマ
Fig.6　Russian-Doll schema

```
<?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"/>
        <xs:element ref="CreditCardNumber"/>
        <xs:element ref="ExpireDate"/>
        <xs:element ref="CardHolderName"/>
      </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>
```

図７　サラミスライス スキーマ
Fig.7　Salami-Slice schema

そのため，スキーマを記述した人や使用したツールにより，一定の書式で書かれている保証がない．既存のスキーマに適用するには，多くの記述方法に適合した変換パターンを準備する必要がある．

暗号化の種類には，大きく分けて以下の３種類がある．

１）MUST その要素が必ず暗号化される事を期待する．

２）MAY1 その要素が暗号化されていても，されていなくてもかまわない．その要素が複数ある場合には，暗号化された物と暗号化されていない物が混在可能．

３）MAY2 その要素が暗号化されていても，されて

いなくてもかまわない．その要素が複数ある場合には，暗号化された物と暗号化されていない物のどちらか一方のみ．

また，それぞれに対してエレメント暗号とコンテント暗号のパターンがある．

9. 変換方法

図５のデータ中の「CreditCardNumber」を MUST（必ず暗号化）に変換する場合についての例を示す．図７のスキーマを対象に，XSLT による変換処理を行う．変換後のスキーマを図８に示す．

```
<?xml version="1.0" encoding="Shift_JIS"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="htt
p://www.xmlconsortium.org/bukai/ouyou/demo/travel" targetNamespace="h
ttp://www.xmlconsortium.org/bukai/ouyou/demo/travel" elementFormDefau
lt="unqualified">
<xsl:import xmlns:xsl="http://www.w3.org/1999/XSL/Transform" namespa
ce="http://www.w3.org/2001/04/xmlenc#" schemaLocation="xenc-schema.x
sd"/>

  <xs:element name="CreditCardInformation">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="CreditCardAuthority"/>
        <xs:element ref="xenc:EncryptedData" xmlns:xenc="http://w
ww.w3.org/2001/04/xmlenc#"/>
        <xs:element ref="ExpireDate"/>
        <xs:element ref="CardHolderName"/>
      </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:element name="CouponIssueDate" type="xs:date"/>
</xs:schema>
```

図８　変換後スキーマ

Fig.8　Post encryption schema

まず，スキーマのルート要素である<xs:schema>とその属性をそのままコピーする．その直下に XML-Encryption のスキーマを import する記述を追加する．XML-Encryption のスキーマは，W3C（World Wide Web Consortium）のサイト上に公開されているので，直接その URL の「http://www.w3.org/TR/2002/REC-xmlenc-core-20021210/xenc-schema.xsd」を指定する．もし，変換後スキーマを利用する場合に直接 Internet にアクセスできなければ，W3C のサイトからスキーマを取得する事ができないため，XML-Encryption のスキーマをローカルに保存しておき，そのファイルを指定しておく．その場合，xenc-schema.xsd でさらに import している XML-Signature のスキーマも同様にローカルに保存し，xenc-schema.xsd 内の schemaLocation を変更しておく．

暗号化対象である要素を置換する．「CreditCardInformation」の中にある「CreditCardNumber」を「xenc:EncryptedData」に置換する必要がある．XSLT で変換するためには，対象の場所を XPath で指定する．「CreditCardInformation」の中にある「CreditCardNumber」は，XPath で「"xs:element[@name='CreditCardInformation']/xs:complexType/xs:sequence/xs:element[@ref='CreditCardNumber']"」と記述する．この要素を「xenc:EncryptedData」と置換する．XSLT のテンプレート機能を使用し，要素数の制限を表す minOccurs と maxOccurs のみを引き継ぐ．

それから，「xenc:EncryptedData」に xenc の名前空間の宣言を追加する．XSLT の仕様として「名前空間宣言は出力されない」となっているので，自由な場所に名前空間宣言を追加することはできない．XSLT ファイルの先頭に xenc の名前空間宣言を行っておくと，xenc 使用時に自動的には名前空間宣言が追加されるので，その機能を利用する．

次に，グローバル宣言の「CreditCardNumber」の宣言を削除する．これはスキーマ中に「CreditCardNumber」が存在していると，「どこかにクレジットカード番号がある」と予想されてしまう事態を防ぐためである．XPath で「"xs:element[@name='CreditCardNumber']"」の要素に対し，何もテンプレートを指定しないことで，変換後スキーマに何も出力されない．

```
<?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">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>

<!-- エレメント暗号、暗号化必須項目 -->
<xsl:template match="xs:element[@name='CreditCardInformation']/xs:complexT
ype/xs:sequence/xs:element[@ref='CreditCardNumber']">
  <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']" />

<!-- それ以外の要素は、コピーする -->
<xsl:template match="@*|node()">
  <xsl:copy>
    <xsl:apply-templates select="@*|node()"/>
  </xsl:copy>
</xsl:template>


<!-- 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>

</xsl:stylesheet>
```

図9　変換用 XSL
Fig.9　XSL for conversion

他要素を暗号化対象にするには，同様に対象箇所の XPath 指定を追加し，相応しいテンプレートを指定する．元スキーマの様々な記述方法に対応するためには，このテンプレートの種類を増やしていく必要がある．

10. 運用上の注意事項

1）元スキーマの公開性

セキュリティを考える上で，元スキーマの公開性についても考慮しておく必要がある．

例えば，クレジットカード番号やパスワード，もしくはさらに機密性の高い情報を暗号化する場合，元スキーマを参照できると「この暗号化されたデータの中に CreditCardNumber がある」ということを中継者が知ってしまうことになる．「非常に価値の高い情報が入っている」という事が分かれば，何らかの攻撃の動機となってしまう可能性があるので，必要であれば元スキーマを公開せず，変換後スキーマのみを中継者に渡す事も考慮する．また，グローバル宣言を使っている場合には，不要になった宣言を削除しておいた方が安全である．

また，NewsML や TravelXML などのように広く一般公開されているスキーマを使う場合は，誰でも元スキーマを入手できる事に注意する．

2）元スキーマと変換後スキーマの同居

元スキーマと変換後スキーマは同じ名前空間であるにも関わらず，異なる構造を持っている．異なる XML ドキュメントは名前空間で区別されるはずが，名前空間で区別できなくなってしまっている．例えば Java のコードを出力した場合には，同じ Package の場所に異なる内容の class を出力しようとし，上書きされてしまう．そのため，同一の環境で，元スキーマと変換後スキーマから自動生成されたものを扱うには，十分に注意する必要がある．異なるルールによる複数の変換後スキーマを扱う場合も同様に注意が必要である．

11. まとめと考察

本稿では，複数サイト間の Web サービスでの中継者において，WS-Security によって部分暗号化された XML ドキュメントを受信し，暗号化されたままの状態で別の Web サービスに対して送信することで End-to-End のセキュリティを実現する方法として，XSLT によって変換したスキーマを使用する方式について述べた．

本方式を検討する上で，「変換後のスキーマの名前空間を変更するかどうか？」は非常に重要である．XML-Encryption は，暗号化前後での名前空間の変更については規定していない．「ある名前空間の指しているスキーマはユニークであること」であるにも関わらず，構造の違うスキーマに同じ名前空間を指定するのは，混乱の原因になりかねない．しかし，6 章で挙げた理由以外にも，既存の WS-Security の実装が名前空間を変更しないなどの理由で，そのままの名前空間を使うこととした．しかし，送信者が名前空間を変えて送ってきた場合には，中継者も変更した名前空間に対応しなければならない．

本稿では扱わなかったが，WS-Security での『署名』は『部分暗号化』とはまったく違う注意が必要である．署名検証のためには，prefix やインデントが完全に同じである必要があるが，スキーマではどちらも規定できない．そのため，中継者でバインディングし，中継先に送信する SOAP メッセージを 1 から作成すると，異なる prefix やインデントを持った SOAP メッセージを作成してしまい，署名検証が失敗する．署名検証を成功させるためには，バインドした各値ではなく，元 SOAP メッセージの該当箇所の XML ドキュメントを使用する必要がある．

今までは，Web サービスと言えばシステム間の通信手段として使われてきた．しかし近年では Ajax の登場により，ブラウザから直接 Web サービスを呼び出すこともできるようになった．ブラウザが WS-Security を使った送信者になることができれば，ブラウザを使っている利用者からサービス提供者までの End-to-End のセキュリティが実現する．中継者は，以前は「使わないかもしれないが，すべてのデータを保存しておく」としていたが，個人情報保護法の影響もあり「不要なデータは，見ない」とした方が良い場面も増えてきている．関係無いデータが参照できない事を保障するためにも，SSL/TLS では不可能だった End-to-End のセキュリティのニーズが高まってくると思われる．その時，サービスを二次利用するサイトはすべて中継者となるため，WS-Security による暗号化・復号化の機能は不要でも，WS-Security を中継する機能が必須となる．

12. 謝辞

実証実験は，XML コンソーシアム応用技術部会とセキュリティ部会の合同で実施した sPlat プロジェクトにおいて検討したものをまとめたものです．実験には，アドソル日進株式会社，株式会社内田洋行，日本電気

株式会社,株式会社ノムラシステムコーポレーション，株式会社日立製作所，PFU アクティブラボ株式会社，富士ゼロックス株式会社，キヤノン株式会社，株式会社ＪＩＥＣ，東京エレクトロン株式会社，株式会社ネット・タイム，富士通株式会社が参加しました．実験に参加された皆さまに心から感謝いたします．

文　献

[1] K.Nakayama, T.Ishizaki, and M.Oba, "Application of Web Services Security Using Travel Industry Model," SAINT 2005 Workshop, pp 358-361, 2005.

[2] 中山弘二郎，植田良一，大場みち子，"XML 暗号による部分暗号化後のスキーマ検証方法，"電気学会 情報システム研究会，IS-06-06，2006.

[3] W3C, "Simple Object Access Protocol (SOAP) 1.1", W3C Note (2002), http://www.w3.org/TR/2000/NOTE-SOAP-20000508/

[4] WS-I: "Security Challenges, Threats and Countermeasures Version 1.0", WS-I Final Material, (2005), http://www.ws-i.org/Profiles/BasicSecurity/SecurityChallenges-1.0.pdf

[5] W3C: "XML Encryption Syntax and Processing", W3C Recommendation (2003), http://www.w3.org/TR/2002/REC-xmlenc-core-20021210/

[6] W3C, "Web Services Description Language (WSDL) 1.1", W3C Note (2001), http://www.w3.org/TR/2001/NOTE-wsdl-20010315

[7] W3C, "XSL Transformations (XSLT) Version 1.0", W3C Recommendation (1999), http://www.w3.org/TR/1999/REC-xslt-19991116

[8] W3C, "XML Schema Part 0: Primer Second Edition", W3C Recommendation (2004), http://www.w3.org/TR/2004/REC-xmlschema-0-20041028/

[9] OASIS, "Web Services Security: SOAP Message Security 1.0 (WS-Security 2004)", OASIS Standard (2004), http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0.pdf

[10] XML コンソーシアム，「TravelXML 利用 Web サービス実証実験プロジェクト 成果 公開資料」(2004), http://www.xmlconsortium.org/koukai/travelxml-p/

[11] XML コンソーシアム，「TravelXML 仕様公開ページ」, http://www.xmlconsortium.org/wg/TravelXML/TravelXML_index.html

[12] XML コンソーシアム，「sPlat プロジェクト プレスリリース」（2006），http://www.xmlconsortium.org/release/pdf/px060406-security-project-final2.pdf