暗号化ツール検証報告書

2010 年 01 月 13 日
XML コンソーシアム
セキュリティ部会 Web サービス実証部会

＜利用条件＞

本書は、本書に記載した要件･技術･方式に関する内容が変更されないこと、および出典を明示いただくことを前提に、無償でその全部または一部を複製、翻案、翻訳、転記、引用、公衆送信等して利用できます。なお、全体を複製、翻案、翻訳された場合は、本書にある著作権表示および利用条件を明示してください。

本書の著作権者は、本書の記載内容に関して、その正確性、商品性、利用目的への適合性等に関して保証するものではなく、特許権、著作権、その他の権利を侵害していないことを保証するものでもありません。本書の利用により生じた損害について、本書の著作権者は、法律上のいかなる責任も負いません。

1. 初めに

XML コンソーシアム セキュリティ部会では、2008 年の MOF2008 (Manufacturing Open Forum 2008)の実証デモシステムにおけるセキュリティについての検討を実施し、その報告書を作成した。[1]
その検討を深める意味で、実際の XML インスタンスを使った複数の暗号化パターンによる暗号化･復号の検証、複数の暗号化ツールにおける暗号化･復号の検証、複数のツール間での復号互換性の確認を実施し、2009 年 5 月 12 日の XML コンソーシアム Week にて検証結果の概要を報告した。[2]
また、検証に必要となるコーディングやテスト作業においては、Web サービス実証部会の協力を得て、共同して検証活動を実施した。
本報告書は、その検証結果の詳細報告書である。

1.1. 検証の目的

2008 年の活動では、Apache XML Security のサンプル･プログラムを利用して、暗号化･復号のテストを実施した。その中で単一の XML データ構造のテストしかしておらず、実際に暗号化を実施する上でのパターンとしては不足している事が課題として残った。
更に暗号化･復号ツールとして、Apache XML Security だけではなく、他の暗号化ライブラリーによる複数のツールでの検証とその互換性検証も追加の課題として残った。本検証では、上記課題を解決することを目的としている。

1.2. 検証期間と参加者

検証作業は 2009 年 2 月から開始し、4 月に完了した。
1 月 9 日－第 6 回部会にて、暗号化ツールの継続検証を決定
2 月 20 日－第 7 回部会にて、Web サービス実証部会との協業を決定
3 月 18 日－第 8 回部会にて、暗号化の要件及びその対象と方式を決定
4 月 16 日－第 9 回部会にて、検証結果確認と報告内容を決定
5 月 12 日－XML コンソーシアム Week にて検証結果を報告

検証メンバーは、以下の通り。(順不同)

- 荒本道隆 (アドソル日進)
- 上村準也 (キヤノンソフト情報システム)
- 大沼啓希 (日本アイ･ビー･エム株式会社)
- 松永豊 (東京エレクトロン デバイス株式会社)

1.3. 検証の範囲

1.1 節で説明しているように今回の検証では、2.2 節の 6 つのパターンの対象要素や構造の暗号化･復号と、それらを各々以下の 3 つのﾗｲﾌﾞﾗﾘｰでの稼動検証及び各ﾗｲﾌﾞﾗﾘｰ間での相互復号検証を実施した。

（1） Apache XML Security

（2） IBM Java SDK 6.0 (以下 IBM Java6 と略す)の JSR-106

（3） Microsoft .NET Framework (以下.NET と略す)

検証には暗号方式として AES+TripleDES を使用した。

AES の鍵の長さは 3 種類(128,192,256Bit)で検証した。

ブロック暗号を使用している AES のパディング方式は W3C の仕様で決まっているが、今回相互復号の検証の為にそれが変更可能であれば変更して検証を行った。

これらの検証だけでも、3 章で説明する 3 つの課題（AES の鍵長、 パディング方式の相違、 KeyName）が浮かび上がり、それらの対応策を検討し適用することができた。

2. 検証方法

2.1. 使用する暗号化ツールのプラットフォーム

以下のソフトウェアを使用した環境下での検証を行った。

2.1.1. Apache XML Security

検証プログラムは Java と C++が提供されているが、Java で検証を実施した。
更に、プログラムは Sun Java を前提としている。

- Windows XP SP3
- Sun JDK 1.6.0.16+JCE (jdk-6u16-windows-i586.exe)
  http://java.sun.com/javase/downloads/index.jsp#need
  Sun Java Platform, Standard Edition 6 SDK は、以下 Sun Java6 と略す
- Apache XML Security Version 1.4.3 (xml-security-bin-1_4_3.zip)
  http://santuario.apache.org/dist/java-library/
- Ant 1.7.1 (apache-ant-1.7.1-bin.zip)
  http://ant.apache.org/bindownload.cgi
- JUnit 4.7 (junit-4.7.zip)
  http://sourceforge.net/projects/junit/files/junit/

2.1.2. JSR-106

検証を実施した 2009 年 4 月時点で、JSR-106 は IBM Java6 にしか実装されていなかったので、WebSphere Application Server Community Edition 2.1.1.3 で提供されている IBM SDK 6 SR5 を使用した。

- Windows XP SP3
- IBM SDK 6 SR5（JCE は同梱 ibm-java-sdk-60-win-i386.exe）
  https://www14.software.ibm.com/webapp/iwm/web/reg/download.do?source=wsced&S_PKG=2113&S_TACT=105AGX90&lang=en_US&cp=UTF-8
  Server and IBM SDK 6 SR5 for Windows の項
  wasce_ibm60sdk_setup-2.1.1.3-ia32win.zip (194 MB)

2.1.3. Microsoft .NET Framework

Microsoft の Visual C# 2008 Express Edition を Web から導入した

- Windows XP SP3
- Microsoft Visual C# 2008 Express Edition
  http://www.microsoft.com/japan/msdn/vstudio/express/

2.1.4. XML インスタンスの比較ツール

- Windows XP SP3
- XMLUnit-1.2 (xmlunit-1.2-bin.zip)
  http://sourceforge.net/projects/xmlunit/files/
- JUnit 4.7 (junit-4.7.zip)
  http://sourceforge.net/projects/junit/files/junit/

2.2. 使用する XML インスタンス

検証に使用する XML インスタンスのサンプルは、XML コンソーシアムが仕様を開発し、標準化を行った ContactXML[3]のスキーマを利用した。

```
<?xml version="1.0" encoding="UTF-8"?>
<ContactXML version="1.1" creator="http://www.pfu.co.jp/meishi-app/xml/contact/1.0"
xmlns="http://www.xmlns.org/2002/ContactXML">
  <ContactXMLItem lastModifiedDate="2009-08-24T14:10:59+9:00">
    <PersonName>
      <PersonNameItem xml:lang="ja-JP">
        <FullName pronunciation="ｽｽﾞｷ ｲﾁﾛｳ">鈴木　一郎</FullName>
        <LastName pronunciation="ｽｽﾞｷ">鈴木</LastName>
        <FirstName pronunciation="ｲﾁﾛｳ">一郎</FirstName>
      </PersonNameItem>
    </PersonName>
    <Occupation>
      <OccupationItem xml:lang="ja-JP">
        <OrganizationName pronunciation="ｼｱﾄﾙ・ﾏﾘﾅｰｽﾞ">シアトル・マリナーズ</OrganizationName>
        <JobTitle>野球選手</JobTitle>
        <Department>一軍</Department>
      </OccupationItem>
    </Occupation>
    <Extension>
      <ExtensionItem xml:lang="ja-JP" extensionType="Extended" name="Seattle Mariners">Seattle
Mariners</ExtensionItem>
    </Extension>
    <Address>
      <AddressItem locationType="Office">
        <AddressCode codeDomain="ZIP">98134</AddressCode>
        <FullAddress xml:lang="ja-JP">1250 First Ave. South Seattle, WA 98134</FullAddress>
        <AddressLine xml:lang="ja-JP" addressLineType="Prefecture">WA</AddressLine>
        <AddressLine xml:lang="ja-JP" addressLineType="City">South Seattle</AddressLine>
        <AddressLine xml:lang="ja-JP" addressLineType="Number">1250 First Ave.</AddressLine>
      </AddressItem>
    </Address>
    <Phone>
      <PhoneItem usage="Official" phoneDevice="Phone">(206) 346-4001</PhoneItem>
      <PhoneItem usage="Official" phoneDevice="Fax">(206) 346-4001</PhoneItem>
    </Phone>
    <Email>
      <EmailItem usage="Official" emailDevice="Unknown">sea@seattle.mariners.mlb.com</EmailItem>
    </Email>
  </ContactXMLItem>
</ContactXML>
```

暗号化対象要素や構造は、W3C, “XML Encryption Syntax and Processing”, W3C

Recommendation 10 December 2002[4]の 2.1 Encrypton Granularity を参考に以下の 6 パターンを選択した。

1) Root ContactXML(要素全体) Encrypton Granularity 2.1.4
2) PersonName+Occupation(複数の構造) Encrypton Granularity 2.1.1
3) ExtensionItem(単一の要素) Encrypton Granularity 2.1.2
4) AddressItem(単一の構造) Encrypton Granularity 2.1.1
5) PhoneItem(複数の要素) Encrypton Granularity 2.1.2
6) EmailItem(テキストのみ) Encrypton Granularity 2.1.3

Encrypton Granularity 2.1.5 で定義されている“暗号化された XML インスタンスの再暗号化”のパターンは検証していない。

2.3. 使用する暗号化ツールの実装

XML 暗号･XML 署名を実践するための Java の標準 API は、それぞれ JSR-106（XML Digital Encryption APIs）と JSR-105（XML Digital Signature APIs）で規定されているが、JSR-106 はまだ仕様策定途中である。今回、Sun Java で使用可能な Apache XML Security を検証に使用し、IBM Java6 には JSR-106 が含まれているので、これを検証に使用した。更に、Microsoft .NET Framework には、XML 暗号･XML 署名の実装が標準で入っているので、これも検証に使用した。

2.3.1. Apache XML Security

Apache XML Security は、Apache Software Foundation の Apache XML プロジェクトによって開発されている XML 暗号･XML 署名の実装である。Apache XML Security は、当該プロジェクトの Web サイト(http://xml.apache.org/security/)により Sun Java および C++の実装が公開されている。以下では Apache XML Security の独自 API を用いた Java 版の実装のコード例について説明する。

Apache XML Security を用いた XML 暗号化のコード例をリスト 1 に示す。

```
1  // 要素を暗号化する AES 鍵の生成
2  KeyGenerator keyGenerator = KeyGenerator.getInstance("AES");
3  keyGenerator.init(128);
4  Key symmetricKey = keyGenerator.generateKey();
5  // AES 鍵を暗号化するための TripleDES 鍵の生成
6  keyGenerator = KeyGenerator.getInstance("DESede");
7  SecretKey kek = keyGenerator.generateKey();
```

```
8   String algorithmURI = XMLCipher.TRIPLEDES_KeyWrap;
9   XMLCipher keyCipher = XMLCipher.getInstance(algorithmURI);
10  keyCipher.init(XMLCipher.WRAP_MODE, kek);
11  //AES 鍵の暗号化
12  EncryptedKey encryptedKey =
        keyCipher.encryptKey(document, symmetricKey);
13  // 要素の暗号化方法の設定
14  XMLCipher xmlCipher = XMLCipher.getInstance(algorithmURI);
15  xmlCipher.init(XMLCipher.ENCRYPT_MODE, symmetricKey);
16  // 暗号化データに組み込む<KeyInfo>要素の設定
17  EncryptedData encryptedData = xmlCipher.getEncryptedData();
18  KeyInfo keyInfo = new KeyInfo(document);
19  keyInfo.add(encryptedKey);
20  encryptedData.setKeyInfo(keyInfo);
21  // 暗号化する要素の取り出し
22  Element enElement =
        (Element) document.getElementsByTagName("要素名").item(int 要素位置);
23  // 暗号化すべきデータを EncryptedData 要素により置換
24  xmlCipher.doFinal(document, enElement, true);
```

リスト 1 Apache XML Security による XML 暗号化の実装

ここで document は、暗号化対象の XML 文書を DOM によりパースした結果のドキュメント（org.w3c.dom.Document インターフェースの実装クラスのインスタンス）である。
2～4 行目では要素を暗号化するための AES 鍵を生成している。6～10 行目では AES 鍵を暗号化するための TripleDES 鍵を生成している。12 行目では AES 鍵の暗号化を行っている。14～15 行目では要素の暗号化方法の設定を行っている。17～20 行目では復号に必要な暗号化データに組み込む<KeyInfo>要素の設定を行っている。22 行目では暗号化する要素を取り出している。
24 行目では暗号化すべきデータを EncryptedData 要素で置換して document に暗号化 XML 文書を完成させる doFinal メソッドを呼び出している。
リスト 1 には含まれていないが、最後に、暗号化済みの document と共通鍵の kek をファイルに書き出す必要がある。

2.3.2. JSR-106

IBM Java6 の JSR-106 実装を利用した検証ツールのコア部分について説明する。
検証ツールはサイト "developerWorks" [5]にて公開されている情報を元に作成した。
上記のサイトで挙げられているサンプルコードはシンプルなものであり、今回の検証で必要とされる、復号処理の結果を元の XML 文書と同じ DOM 構造へ復元するような処理は含んでいない。(暗号化されていた要素のコンテンツ部分を復号してコンソール出力するようになっている)
また、暗号化に関しても多様なパターンを処理するものではなかった。
結果、今回の検証の手順で定めた6つのパターンを処理可能とするため、検証ツールは独自のコード(特に DOM の操作)を多く含むものとなった。
そのため、検証ツールの実装担当者の誤解や間違いの含まれる可能性も高い。この点を理解いただきたい。

(1) 復号

以下、JSR-106 による復号の概念のコード例をリスト 2 に示す。このコードは完全ではなく、説明の為に提示するものである。

```
1  public Document decrypt(Document doc, Key kek) throws Exception {
2          KeySelector selector = new SimpleKeySelector(kek); ･･･①
3          NodeList elements = doc.getElementsByTagNameNS(EncryptedData.XMLNS,"EncryptedData");
4          int index = elements.getLength();
5          while (index-- > 0) {  ･･･②
6                Element element = (Element) elements.item(index);
7                DOMDecryptContext context = new DOMDecryptContext(selector, element);  ･･･③
8                EncryptedType data = xf.unmarshalEncryptedType(context);
9               decryptAndReplace((EncryptedData) data, context, doc);  ･･･④
10         }
11         return doc;
12 }
```

リスト 2 JSR-106 による XML 復号の実装

①KeySelector インターフェースの実装である SimpleKeySelector クラスのインスタンスを作成
これは復号に使われる鍵を選択するために DOMDecryptContext によって使われる。

今回の検証では、暗号化された要素の子要素として、その鍵情報が含まれる形となっているため、それに対応するごくシンプルな SimpleKeySelector クラスを独自に実装した。

②XML 文書中の暗号化された要素を逆順に復号するループ
復号中、XML 文書中の暗号化されていた要素が復号された要素で置換されるため、DOM ツリーが変化する。
これにより 2 つ以上の要素が復号される場合でも、連続して処理するための配慮である。

③DOMDecryptContext のインスタンスを作成し、復号を行う
ここで xf は XMLEncryptionFactory のインスタンスである。
XMLEncryptionFactory#unmarshalEncryptedType() によって EncryptedType のインスタンスが得られる。このインスタンスは、EncryptedData へキャストして使われる。

④ EncryptedData#decryptAndReplace() によって復号と要素の置換を同時に行う
本来はそのように考えられるが、今回の検証の手順で定めた6つのパターンすべてを、同様に処理することはできなかった。
そのため、同名のメソッドを独自に実装し同等の処理を行うようにした。

EncryptedData#decryptAndReplace() の独自の実装を行った理由は以下のとおり。
1. XML 文書のルート要素が暗号化されていた場合、要素の置換に失敗する
2. 復号した後の XML 文書の部分に対する文字エンコーディングの解釈が、プラットフォームに依存する

上記の 1. に関しては、復号だけではなく暗号化でも同様である。どちらも、要素の置換の方法がルート要素である場合を特に考慮していないように思われる。
上記の 2. は復号時のみの問題で、今回の検証では、文字エンコーディングに関する特別な指定がない状況ではあるが、その場合 UTF-8 と解釈するのが妥当のはずである。
しかし、動作させてみると、Java のデフォルトの文字エンコーディングが UTF-8 である Ubuntu 等では正しく復号されたが、Windows XP 等では正しく復号されなかった。

EncryptedData#decryptAndReplace() の独自の実装の内容は以下のとおり。

1. EncryptedData#decryptAndReplace() ではなく、EncryptedData#decrypt() を呼び出し、復号されたデータを読み出すストリームを得る
2. ストリームから、データを文字エンコーディング UTF-8 として読み出し、そのテキスト

から DocumentFragment を作成する
3. 作成した DocumentFragment を XML 文書にインポートし、暗号化されている要素と置換（ルート要素の場合は Node#replaceChild() ではなく Node#removeChild() と Node#appendChild() の組み合わせで置換処理している）

上記の 1. ～ 3. のうち、JSR-106 の API と関連するのは 1. の処理のみ。他は一般的な DOM 操作の処理である。

（2） 暗号化

以下、JSR-106 による暗号化の概念のコード例をリスト 3 に示す。このコードは完全ではなく、説明の為に提示するものである。

```
1  public Document encrypt(Document doc, Key kek, boolean content, List«Element» elements) throws Exception {
2      int index = elements.size();
3      while (index-- > 0) {  ･･･①
4          Element element = elements.get(index);
5          ToBeEncrypted toBeEncrypted;  ･･･②
6          if (content) {
7              toBeEncrypted = new DOMToBeEncryptedXML(element.getChildNodes(), null);
8          } else {
9              toBeEncrypted = new DOMToBeEncryptedXML(element, null);
10         }
11         XMLEncryptContext context;  ･･･③
12         if (parent instanceof Document) {
13             context = new DOMEncryptContext(encryptionKey, element);
14         } else {
15             context = new DOMEncryptContext(encryptionKey, parent, sibling);
16         }
17         EncryptedData encryptedData = xf.newEncryptedData(toBeEncrypted, encryptionMethod, null, null, null);  ･･･④
18         encryptedData.encrypt(context);
19          // 以下、DOM 操作が続くが省略  ･･･⑤
20     }
21     return doc;
```

22	}

リスト 3 JSR-106 による XML 暗号化の実装

① XML 文書中の指定された要素を逆順に暗号化するループ
elements には、暗号化すべき doc の中の子要素があらかじめ指定されているものとする。暗号化の処理中、XML 文書中の元の要素が暗号化された要素で置換されるため、DOM ツリーが変化する。これにより 2 つ以上の要素が暗号化される場合でも、連続して処理するための配慮である。

② DOMToBeEncryptedXML のインスタンスを作成
DOMToBeEncryptedXML のインスタンスには暗号化すべき要素の情報を含ませるが、要素のコンテンツ部分を暗号化するか、要素そのものを暗号化するかにより作成方法が異なる。

③ DOMDecryptContext のインスタンスを作成
ここで encryptionKey は SecretKey のインスタンスであり、暗号化に使われる鍵とする。この後の EncryptedData#encrypt() により、暗号化された要素が XML 文書のどの位置に挿入されるかが、このインスタンスの作成方法により異なる。
ルート要素そのものを暗号化する場合、parent ノードは XML 文書そのものとなるが、そのような指定を行うと、暗号化された要素の挿入時に DOM 操作で失敗してしまう。
そこで、本来の意図とは異なるが、この検証ツールでは、仮に暗号化済みの要素をルート要素の子要素として追加させるようにしている。

④ EncryptedData のインスタンスを作成し、EncryptedData#encrypt() によって暗号化と要素の挿入を同時に行う
ここで xf は XMLEncryptionFactory のインスタンスである。
また、encryptionMethod は EncryptionMethod のインスタンスであり、
XMLEncryptionFactory#newEncryptionMethod() で作成されたものである。
XMLEncryptionFactory#newEncryptedData() によって EncryptedData のインスタンスが得られる。
暗号化は EncryptedData#encrypt() で行う。復号とは異なり、DOM 操作を伴わない（データの暗号化だけを行う）メソッドは用意されていない。
また、暗号化した元の要素は DOM ツリーから取り除かれない。暗号化した要素との置換ではなく、暗号化した要素の挿入のみが行われる。

⑤ 暗号化された元の要素を取り除き、XML 文書を暗号化された形に整える
一般的な DOM 操作により、暗号化された元の要素を取り除く。
この際、ルート要素のそのものを暗号化した場合と、その他の場合では取り除く処理が異なる。
また、暗号化に使った鍵情報を各要素に挿入する必要があり、これも場合に応じて適当な位置に対して行う。
この部分は JSR-106 の API と直接関連しない。但し、鍵情報そのものの作成は JSR-106 の API を使用して行っている。

2.3.3. Microsoft .NET Framework

Microsoft .NET Framework には、XML 暗号・XML 署名の実装が標準で入っており、何も追加しなくても XML 暗号が行える。
Microsoft .NET Framework を用いた C#による XML 暗号化のコード例をリスト 4 に示す。
今回は C#で記述したが、VB.NET や VC++.NET などでも同様に記述できる。

```
1           // 対象の XML ファイルをロードし、暗号化要素を特定する
2           XmlDocument xmlDoc = new XmlDocument();
3           xmlDoc.Load("sample.xml");
4           XmlElement element =
5   (XmlElement)xmlDoc.GetElementsByTagName("ExtensionItem").Item(0);
6           // KEK（TripleDES）を作成する
7           SymmetricAlgorithm kek = new TripleDESCryptoServiceProvider();
8           kek.GenerateKey();
9           // 暗号化をおこなう
10          EncryptedXml xmlEnc = new EncryptedXml(xmlDoc);
11          xmlEnc.AddKeyNameMapping("Alice", kek);
12          EncryptedData encXml = xmlEnc.Encrypt(element, "Alice");
13          EncryptedXml.ReplaceElement(element, encXml, false);
14          // 結果を XML ファイルに保存する
            xmlDoc.Save("aaa.xml");
```

リスト 4 Microsoft .NET Framework を用いた C#による XML 暗号化の実装

2～3 行目で XML 文章を読み込んで、4 行目で暗号化対象の要素を指定する。6～7 行目で TripleDES による鍵を作成しており、ここでは記述していないが復号時に同じ鍵が必要なので、ファイルに書き出しておく必要がある。9～12 行目で XML 暗号化をおこなっている。10 行目で KeyName と TripleDES 鍵の関連付けを行い、11 行目で暗号化対象要素と KeyName の関連付けを行って、12 行目で暗号化をおこなっている。ReplaceElement()の第3引数に false を指定すると、タグ名ごと暗号化される。コンテントのみを暗号化したい（タグ名はそのまま残す）場合は true を指定する。そして 14 行目で、結果をファイルに保存する。

上記のコードで明示していない部分は.NET のデフォルト設定が使用されるため、AES(256)+TripleDES、Padding（ISO10126）で暗号化される。AES の鍵長や Padding を指定したい場合には、9～12 行目の代わりに以下のようなコードを記述する必要がある。

```
1  string encAlgorithmURI= EncryptedXml.XmlEncAES256Url;
2  PaddingMode encPaddingMode = PaddingMode.ISO10126;
3  Boolean isConentEnc = false;
4  EncryptedXml eXml = new EncryptedXml();
5  RijndaelManaged sessionKey = new RijndaelManaged();
6  switch (encAlgorithmURI)
7  {
8  case EncryptedXml.XmlEncAES128Url: sessionKey.KeySize = 128; break;
9  case EncryptedXml.XmlEncAES192Url: sessionKey.KeySize = 192; break;
10 case EncryptedXml.XmlEncAES256Url: sessionKey.KeySize = 256; break;
11 }
12 byte[] encryptedElement = eXml.EncryptData(element, sessionKey, isConentEnc);
13 EncryptedData edElement = new EncryptedData();
14 edElement.Type = EncryptedXml.XmlEncElementUrl;
15 edElement.EncryptionMethod = new EncryptionMethod(encAlgorithmURI);
16 EncryptedKey ek = new EncryptedKey();
17 byte[] encryptedKey = EncryptedXml.EncryptKey(sessionKey.Key, kek);
18 ek.CipherData = new CipherData(encryptedKey);
19 ek.EncryptionMethod = new EncryptionMethod(EncryptedXml.XmlEncTripleDESKeyWrapUrl);
20 edElement.KeyInfo = new KeyInfo();
21 KeyInfoName kin = new KeyInfoName();
22 kin.Value = "Alice";
23 ek.KeyInfo.AddClause(kin);
```

```
24  edElement.KeyInfo.AddClause(new KeyInfoEncryptedKey(ek));
25  edElement.CipherData.CipherValue = encryptedElement;
26  EncryptedXml.ReplaceElement(element, edElement, isConentEnc);
```

リスト 5 AES の鍵長や Padding を指定したい場合の XML 暗号化の実装

デフォルト設定を使うのであれば、他の実装と比較すると短いコード量で済むというのが長所である。
ちなみに、対象の XML 文章と TripleDES 鍵があれば、以下のコードで復号できる。

```
1  EncryptedXml encXml = new EncryptedXml(xmlDoc);
2  encXml.AddKeyNameMapping("Alice", kek);
3  encXml.DecryptDocument();
```

リスト 6 Microsoft .NET Framework を用いた C#によるデフォルト設定の XML 復号の実装

暗号化方式などは暗号結果の XML 内に記述されているので、ここで指定する必要はない。EncryptedData のうち、復号対象の要素を KeyName によって指定し、TripleDES 鍵を渡すだけで復号できる。暗号化の時と同様に、xmlDoc 内の KeyName で指定した箇所が復号されるので、xmlDoc をファイルに保存する必要がある。

2.3.4. 比較ツール

人手に依らず、検証プログラムの出力をチェックする比較ツールを用意した。
比較ツールは XMLUnit v1.2 をそのまま利用したシンプルなものを、今回の検証の手順に合わせて作成している。
比較ツールは以下の動作を行う。
1. 検証ツールによって復号された XML 文書（ファイル名が 'd' で始まるもの）すべてを、暗号化される前のサンプル XML 文書（sample.xml）と比較する
2. 検証ツールによって暗号化された XML 文書（ファイル名が 'e' で始まるもの）すべてについて、EncryptedData 要素が含まれることを確認する

復号された XML 文書は、元の文書と内容が一致しなければならない。
動作 1. には XMLUnit の「XML 文書の比較機能（DetailedDiff クラス）」を利用し、比較ツールは各 XML 文書毎に、違いがなければ OK、違いが 1 箇所でもあれば NG

を出力する。

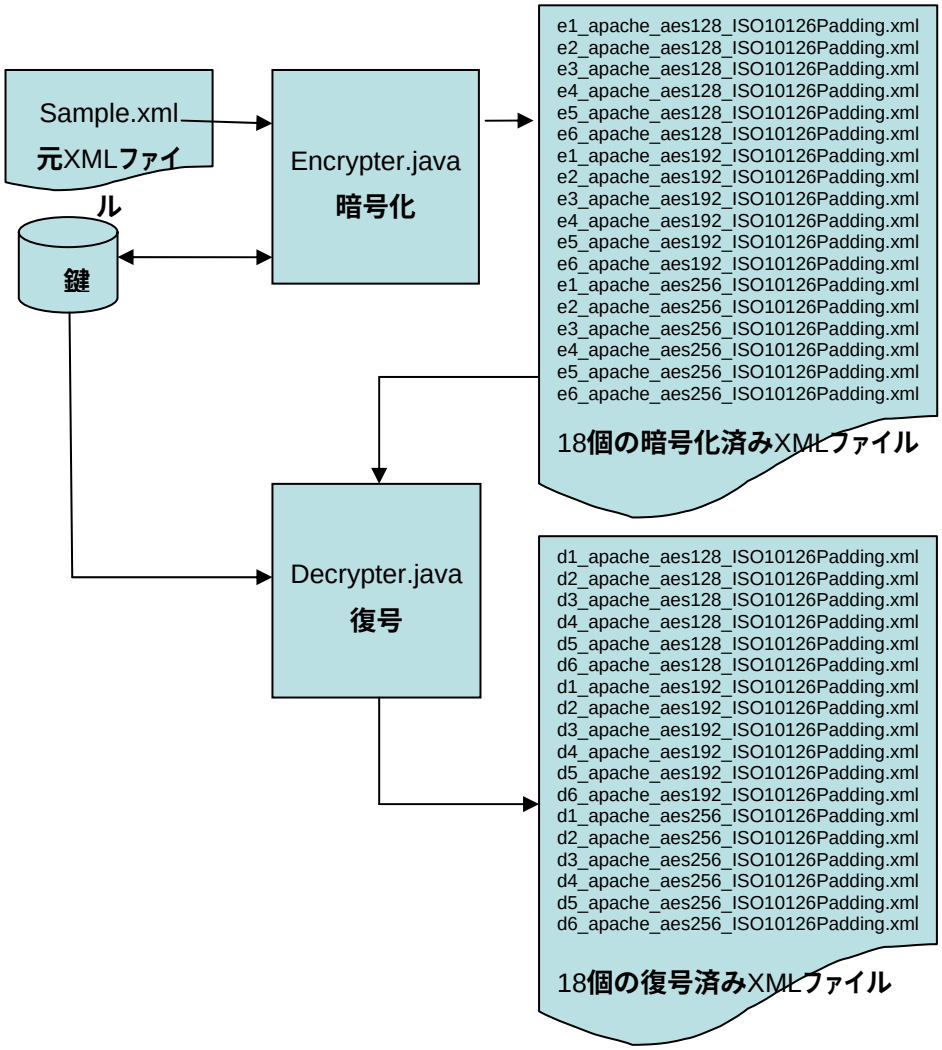
暗号化された XML 文書は、今回の検証の手順で定めた6つのパターンの EncryptedData 要素が含まれなければならない。
動作 2. には XMLUnit の「XML 文書の検証機能（XMLAssert クラス）」を利用し、比較ツールは各 XML 文書毎に、EncryptedData 要素が含まれていれば OK、含まれていなければ NG を出力する。

なお、EncryptedData 要素が含まれるかどうかは、XMLAssert クラスに対し XPath で指定し、指定した要素が存在するかを確認させる。
今回の検証の手順で定めた6つのパターンに対応する XPath は以下のとおりで、比較ツールは XML 文書毎に以下を使い分ける。
（識別子 xenc、xcon はそれぞれ名前空間 http://www.w3.org/2001/04/xmlenc#、http://www.xmlns.org/2002/ContactXML を表す）

1. /xenc:EncryptedData
2. /xcon:ContactXML/xcon:ContactXMLItem/xenc:EncryptedData[1] および xenc:EncryptedData[2]
3. /xcon:ContactXML/xcon:ContactXMLItem/xcon:Extension/xenc:EncryptedData
4. /xcon:ContactXML/xcon:ContactXMLItem/xcon:Address/xenc:EncryptedData
5. /xcon:ContactXML/xcon:ContactXMLItem/xcon:Phone/xenc:EncryptedData[1] および xenc:EncryptedData[2]
6. /xcon:ContactXML/xcon:ContactXMLItem/xcon:Email/xcon:EmailItem/xenc:EncryptedData

2.4. 検証プログラム

2.4.1. 検証プログラムのフロー

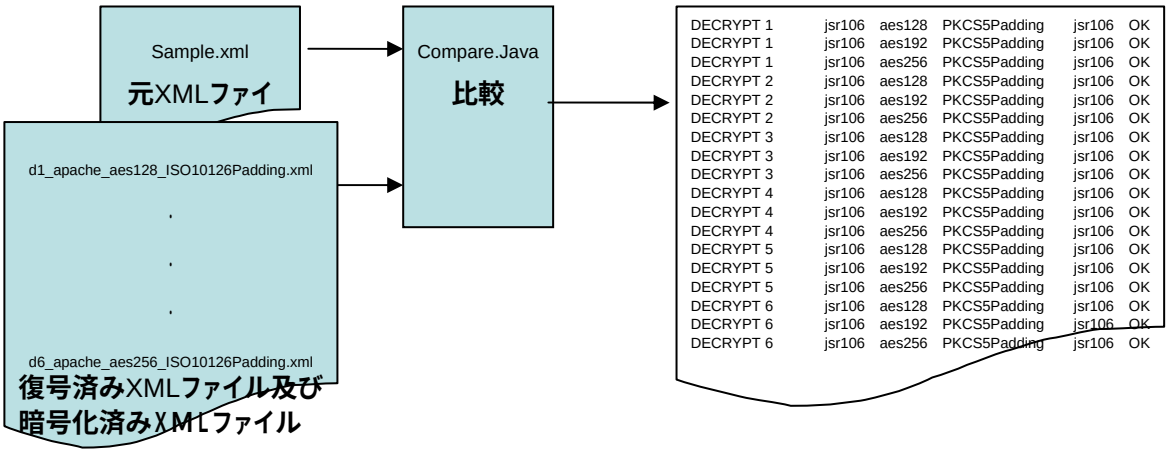


2.4.2. 元文書との比較ツール

元の Sample.xml と復号された XML インスタンスとの比較及び暗号化済み XML ファイルの<EncryptedData>要素の存在チェックを行う。

元の Sample.xml と復号された XML インスタンスが一致していると“OK”と表示する。

更に暗号化済み XML ファイルの<EncryptedData>要素の存在を確認すると“OK”と表示する。



2.5. 検証手順

2.5.1. Apache XML Security による暗号化・復号検証

以下の手順で暗号化 XML ファイル（36 個）、復号 XML ファイル(36 個)を作成する。

(1) 暗号化1：標準の Padding(ISO10126)で暗号化し、暗号化ファイル(6 パターン x3 つの AES 鍵長なので 18 個)を作成

(2) 復号1：(1)の 18 ファイルを復号して、復号ファイル(18 個)を作成

(3) 暗号化 2：Padding を PKCS5 に変更して暗号化し、暗号化ファイル(18 個)を作成

(4) 復号 2：(3)を復号して、復号ファイル(18 個)を作成

(5) 比較：(2)(4)の 36 個のファイルを元の Sample.xml と比較

2.5.2. JSR-106 による暗号化・復号検証

以下の手順で暗号化 XML ファイル（18 個）、復号 XML ファイル(18 個)を作成する。

(6) 暗号化：暗号化（Padding 方式は PKCS5)し、暗号化ファイル(18 個)を作成

(7) 復号：(6)を復号して、復号ファイル(18 個)を作成

(8) 比較：(7)の 18 個のファイルを元の Sample.xml と比較

2.5.3. Microsoft .NET Framework による暗号化・復号検証

以下の手順で暗号化 XML ファイル（36 個）、復号 XML ファイル(36 個)を作成する。

(9) 暗号化：W3C 標準の Padding(ISO10126)と標準で無い Padding(PKCS5)で暗号化し、暗号化ファイル(36 個)を作成

(10)復号：(9)を復号して、復号ファイル(36 個)を作成

(11)比較：(10)の 36 個のファイルを元の Sample.xml と比較

2.5.4. Apache XML Security による暗号化ファイルを JSR-106 で復号

(12)復号：(1)の内 Padding(PKCS5)の 18 ファイルを復号して、復号ファイル(18 個)を作

成

(13)比較：(12)の 18 ファイルを元の Sample.xml と比較

2.5.5. Apache XML Security による暗号化ファイルを.NET で復号

(14)復号：(1)(3)を復号して、復号ファイル(36 個)を作成

(15)比較：(12)の 36 ファイルを元の Sample.xml と比較

2.5.6. JSR-106 による暗号化ファイルを Apache XML Security で復号

(16)復号：(6)を復号して、復号ファイル(18 個)を作成

(17)比較：(16)の 18 ファイルを元の Sample.xml と比較

2.5.7. JSR-106 による暗号化ファイルを.NET で復号

(18)復号：(6)を復号して、復号ファイル(18 個)を作成

(19)比較：(18)の 18 ファイルを元の Sample.xml と比較

2.5.8. .NET による暗号化ファイルを Apache XML Security で復号

(20)復号：(9)を復号して、復号ファイル(36 個)を作成

(21)比較：(20)の 36 ファイルを元の Sample.xml と比較

2.5.9. .NET による暗号化ファイルを JSR-106 で復号

(22)復号：(9)の内 Padding(PKCS5)の 18 ファイルを復号して、復号ファイル(18 個)を作成

(23)比較：(22)の 18 ファイルを元の Sample.xml と比較

3. 検証結果

検証した結果と起こった課題及びその対応策を説明する。

3.1. 結果サマリー

3.1.1. 初期状態

初期状態では、以下の結果が得られた。鍵長が 192 及び 256 ビットの場合、Apache XML Security と JSR-106 では暗号化が Illegal key size or default parameters となり、同様に復号でも、Apache XML Security と JSR-106 で同じエラーとなった。
また、JSR-106 では、他の方法で暗号化された XML ファイルは復号できなかった。

		復号		
暗号化	AES 鍵長	Apache	JSR-106	.NET
Apache	128	○	×	○
Apache	192	NA	NA	NA
Apache	256	NA	NA	NA
JSR-106	128	×	○	×
JSR-106	192	NA	NA	NA
JSR-106	256	NA	NA	NA
.NET	128	○	×	○
.NET	192	NG	NG	○
.NET	256	NG	NG	○

○:復号 OK
×:復号 NG　　Given final block not properly padded
NG:復号 NG　　Illegal key size or default parameters
NA:暗号化 NG　　Illegal key size or default parameters

3.1.2. 最大暗号化強度の制限を解除

Apache XML Security と JSR-106 でも鍵長が 192 及び 256 ビットの場合に暗号化できるように、以下のサイトから最大暗号化強度の制限を解除してあるポリシーファイルをダウンロードし、それに置換した所、以下の結果が得られた。
サイト 1：Sun "Java Cryptography Extension (JCE) Unlimited Strength Jurisdiction Policy Files 6"[6]
サイト 2：IBM SDK Policy files[7]

		復号		
暗号化	AES 鍵長	Apache	JSR-106	.NET
Apache	128	○	×	○
Apache	192	○	×	○
Apache	256	○	×	○
JSR-106	128	×	○	×
JSR-106	192	×	○	×
JSR-106	256	×	○	×
.NET	128	○	×	○
.NET	192	○	×	○
.NET	256	○	×	○

○:復号 OK

×:復号 NG　　Given final block not properly padded

JSR-106 では、他の方法で暗号化された XML ファイルは復号できなかったし、JSR-106 で暗号化した XML ファイルは、他の方法で復号できなかった。
それ故 Given final block not properly padded というエラーメッセージからパディング方式が異なるのが理由と考えられた。

3.1.3. Padding の ISO10126 から PKCS5 への変更（Apache,.NET）

Apache XML Security と Microsoft .NET Framework に対して、Padding 方式を ISO10126 と PKCS5 の両方を使用できるようにしたところ、3 種類の鍵長において以下の結果を得た。これにより、IBM Java6 の JSR-106 は、PKCS5 のパディング方式しか扱えないことが判明した。

		復号		
暗号化	Padding	Apache	JSR-106	.NET
Apache	ISO10126	○	×	○
Apache	PKCS5	○	○	○
JSR-106	PKCS5	○	○	○
.NET	ISO10126	○	×	○
.NET	PKCS5	○	○	○

○:復号 OK

×:復号 NG　　Given final block not properly padded

3.2. AES 鍵長の課題

Sun Java6 と IBM SDK 6 は、どちらも AES で使用できるビット数が 128 までである。
XML 暗号の W3C 勧告によれば AES-256 は必須（REQUIRED）なので、Sun Java6 と IBM Java6 にそれぞれ JCE（Java Cryptography Extension）をインストールして最大暗号化強度の制限を解除し、AES-256 を使用可能にする必要がある。.NET ではデフォルトで AES-256 が使用可能になっている。

3.3. パディング方式の相違の課題

今回の検証では、IBM Java6 の JSR-106 実装が、他の実装によって暗号化された XML 文書を復号できない場合があることが判明した。
その際、以下の Java 例外が発生している。

```
javax.crypto.BadPaddingException: Given final block not properly padded
  at com.ibm.xml.enc.dom.DOMEncryptedData.decrypt(DOMEncryptedData.java:510)
```

この例外の情報を元に、問題はパディング形式の違いによるものと推測した。
仮説として「他の実装は ISO10126 形式に従ったデータを出力しているが、これを IBM Java6 の JSR-106 実装は PKCS5 形式で解釈している」を立てた。

ISO10126 形式は W3C 「XML Encryption Syntax and Processing」内で定義されたもので、XML 文書の暗号化はこの形式に従うのが正しい。
PKCS5 形式は PKCS#5 で使用されたもので、RFC 1423 としても知られており、良く普及している形式だが、本来、XML 文書の暗号化には用いない。

仮説の検証として以下の手順を試みた。

(1) サンプル XML 文書（sample.xml）の一部を以下のように書き換え、コンテンツ部分が 63 バイトとなるようにした。

```
<EmailItem emailDevice="Unknown"
usage="Official">123456789012345678901234567890123456789012345678901234567
890123</EmailItem>
```

(2) 書き換えた XML 文書の EmailItem 要素だけを他の実装によって暗号化し、IBM Java6 の JSR-106 実装で復号した。

この手順で行えば、復号処理に成功することを確認した。これは、コンテンツの長さが 62 バイトの時には復号できないが、コンテンツ部分を 63 バイトに調整することで、パディングの形式に関わらず結果的に同じデータが復号され、PKCS5 形式としても解釈可能となって失敗しないと考えられるのでパディングの課題と言うことが証明された。

なお、ISO10126 と PKCS5 の2つの形式で同じデータが出力されるのは、どちらも最後のバイトにパディング長を入れる手法が共通しているためである。
パディングとは、ブロック暗号処理の際に、暗号化するデータがブロック長の整数倍となるよう、データにダミーのバイトをいくつか追加しておくという操作である。
ダミーであるため、復号処理の際に、元のデータだけを正しく取り出すには、パディングの部分を判別できるよう一定の形式に従う必要がある。

例えば、暗号化に際して 1, 2, 3, 4 バイトのパディングが必要になった場合、ISO10126 形式では以下の様に、元のデータの最後にダミーのバイトを付加する。

データ	データ	データ	データ	0x01
データ	データ	データ	0x??	0x02
データ	データ	0x??	0x??	0x03
データ	0x??	0x??	0x??	0x04

0x?? はどのような値でも構わない。この値は実装によって異なると思われるが、多くは 0x00 を使用している。

PKCS5 形式でのパディングで以下の様になる。

データ	データ	データ	データ	0x01
データ	データ	データ	0x02	0x02
データ	データ	0x03	0x03	0x03
データ	0x04	0x04	0x04	0x04

コンテンツの長さが 63 バイトの場合にはパディング部分が 1 バイトになり、ISO10126 と PKCS5 の2つの形式でパディングを含めた結果が同じとなり、パディング形式による違いが無くなるため、正しく復号できる。

パディング部分が 2 バイト以上の場合は、ISO10126 形式のものを PKCS5 として見ると、最後のバイト以外が異なっているために正しくないと判断され、復号できない。

ただし、PKCS5 形式のものを ISO10126 として見ると、最後のバイト以外は無視するために正しいとみなされるので、IBM Java6 の JSR-106 で暗号化した出力は Apache XML Security と Microsoft .NET Framework で復号できる。

3.4. KeyName の課題

暗号化した XML に「KeyInfo/EncryptedKey/KeyInfo/KeyName」という要素は必須ではないが、今回の検証では.NET での復号のコードを簡略化するために、KeyName 要素を付けた形式で行った。

4. 終わりに

この検証は、XMLインスタンスの 6 種の暗号化パターンと複数の暗号化ツールの利用が当初のゴールであった。容易にこれらの検証が完了できので、相互復号検証を加えたところ、3 つの課題が浮かび上がった。 それらは、比較的短期間にこれらの課題を全て解決することができた。 しかしながら、全ての利用パターンや環境を検証してはいない。例えば、暗号化済みの XML インスタンスの再暗号化、複数キーによる別要素･別構造の暗号化やその復号処理順序、複数 OS 環境下での複数暗号化ライブラリーの利用、複数の鍵の受け渡し（交換）方法や鍵の管理、SOAP への適用(WS-Security)、処理性能やコーディング量の比較･･･等のようなカバーしていない課題がまだ積み残されている。

5. 参考文献

[1] MOF2008 合同デモシステム向けセキュリティ報告書
http://www.xmlconsortium.org/public_doc/mof2008_security/mof2008.html
[2] 第 8 回 XML コンソーシアム Weekー テーマ：XML が支えるエンタープライズシステム新潮流
http://www.xmlconsortium.org/seminar09/090512-13+19-20/090512-prog.html
[3] 『ContactXML 部会』 の活動についてのご紹介
http://www.xmlconsortium.org/bukai/contact_kastudou.html
[4] W3C XML Encryption Syntax and Processing
http://www.w3.org/TR/xmlenc-core/
[5]IBM developerWorks Java XML Digital Encryption API Specification (JSR 106)
http://www.ibm.com/developerworks/java/jdk/security/60/secguides/xmlsecDocs/overview.html
[6] Sun Developer Network Java SE Downloads
http://java.sun.com/javase/downloads/index.jsp
[7] IBM developerWorks Security information
http://www.ibm.com/developerworks/java/jdk/security/60/