署名ツール検証報告書

2010 年 01 月 27 日
XML コンソーシアム
セキュリティ部会

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 (Manufactureing 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、複数の XML 署名方式に対して、複数の XML 署名ライブラリー（API）を使った署名ツールにおける署名と検証のテスト、複数の署名ライブラリー（署名ツール）間での相互運用性の確認の為の検証を実施し、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]
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、Sun Java Platform, Standard Edition 6 JDK （以後 Sun Java6） [3] と Microsoft .NET Framework （以後 .NET） のサンプル･プログラム（署名ツール）を利用して、署名･検証のテストを実施した。その際に XML 署名方式のうち名前空間無しの Enveloping のテストにおいて Sun Java6 と.NET の間で結果が異なる問題を見つけており、この解決または問題点の明確化が課題として残っていた。この課題に対して行った内容の調査結果と、更に Sun Java6 と.NET 以外の署名ライブラリーとして、IBM Java SDK 6.0 （以後 IBM Java6） [4] およびオープンソースの XML Security Library [5] （以後 XMLSec） による追検証結果により問題点を明確化することを目的とした。また 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と重複するが、その時に目的していた以下の項目についても 4 節に付録として再度まとめた。

- XML 署名で一般的な各署名方式による相互運用性の確認
- SHA-2（SHA-256/384/512）への対応状況の確認
- 署名で重要な XML 正規化（C14N）の単独利用の確認
- PKCS#12 ファイルや Windows 証明書ストアの利用方法確認
- その他 XML 署名の基本的なオプションへの対応状況を確認

相互運用性に関しては商用ライブラリーが多い XML 長期署名（XAdES）では ECOM（次世代電子商取引推進協議会）で各社参加して試験（プラグテスト[6]）が行われている。しかし基本となる XML 署名に関しては標準の OS やシステムに含まれて提供される API と言うこともあり、あまり確認がされていない。この為に標準的な XML 署名 API 間の相互運用性の確認は意味があると考えている。

1.2. 検証期間と参加者

検証作業は 2009 年 3 月から開始し、11 月に完了した。

3 月 18 日－2008 年度第 8 回部会にて、署名ツールの継続検証を決定

5 月 12 日－XML コンソーシアム Week にて検証結果を報告

10 月 7 日－2009 年度第 5 回部会にて、XMLSec による追検証を報告

11 月 6 日－ツール検証 ML にて、IBM Java6 による追検証を報告

検証メンバーは、以下の通り。

- 宮地直人 （有限会社ラング･エッジ）
- 小森谷哲 （有限会社ラング･エッジ）
- 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 XML コンソーシアム Week までの検証では、2.2 節の4つのパターン署名方式に対する署名･検証と、それらを各々2 つの署名ライブラリー（Sun Java6 と.NET）での稼動検証及び各ライブラリー間での相互運用性検証のみを実施した。この結果一部 XML 署名方式において署名ライブラリー間の不一致が見つかった。

この為に今回は追加した署名ライブラリー（IBM Java6 と XMLSec）において、Sun Java6 と.NET で作成した署名ファイルへの検証のみを追加実施している。これにより問題点はある程度明確化できたと考えている。今回は署名には生成した秘密鍵か、X.509 証明書と秘密鍵のペアを利用したが、証明書を使った場合でも認証パス検証は検証の対象範囲外とした。

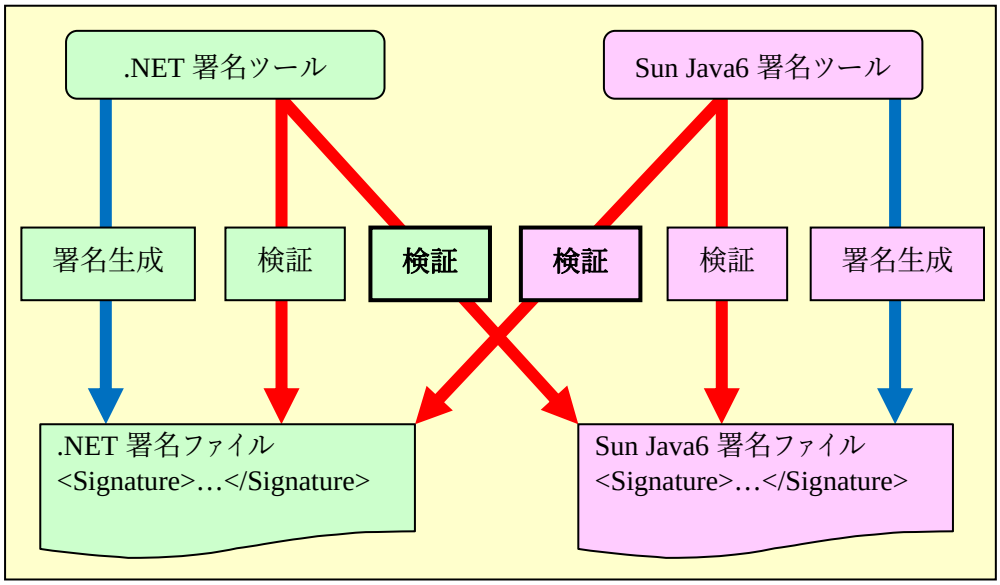


図 1. 検証の概要

2. 検証方法

2.1. 使用する XML 署名ツールのプラットフォーム

以下の環境下で検証を行った。

2.1.1. Java6

- Windows XP SP3 / Sun JDK 6 Update 13
- CentOS 5.3 / Sun JDK 6 Update 17 （検証のみ）
- CentOS 5.3 / IBM JDK 6 （検証のみ）

2.1.2. .NET Framework

- Windows XP SP3
- .NET Framwork 3.5 （要求は.NET Framework 2.0 以上）
- Microsoft Visual C# 2005 Professional Edition

2.1.3. XMLSec

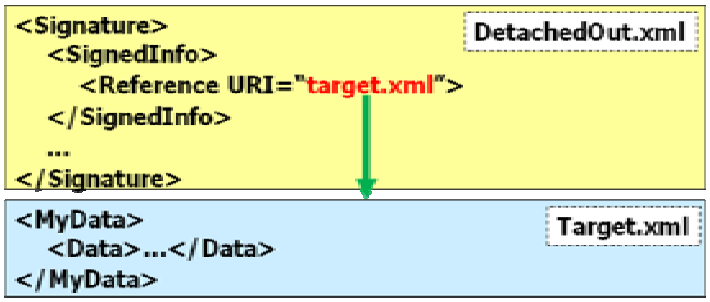
- オンライン検証 http://www.aleksey.com/xmlsec/xmldsig-verifier.html

2.2. 使用する XML 署名方式

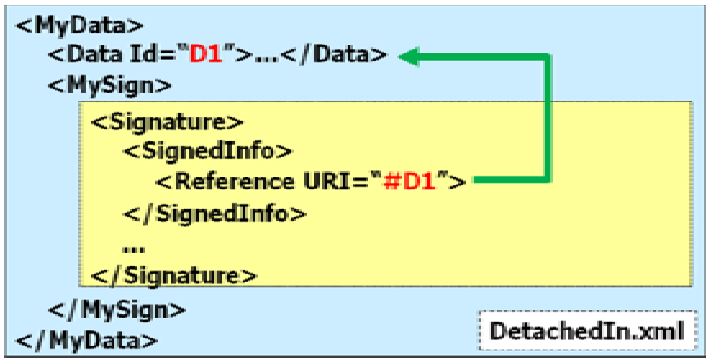
検証に使用する XML 署名[7]方式としては、標準的な以下の 4 パターンとした。一般的に利用される XML 署名方式はこの 4 パターンのいずれかになると考えている。

1） 外部別ファイルへの Detached（外包）
2） 同一ファイル内 XML 要素への Detached（外包）
3） 同一ファイルへの Enveloped（埋め込み）
4） XML 署名内オブジェクトへの Enveloping（内包）

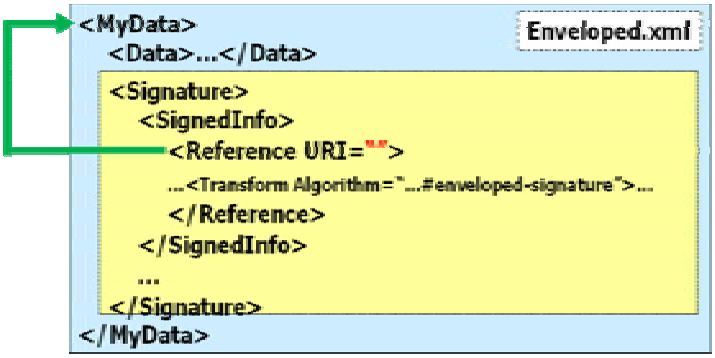
2.2.1. 外部別ファイルへの Detached（外包）



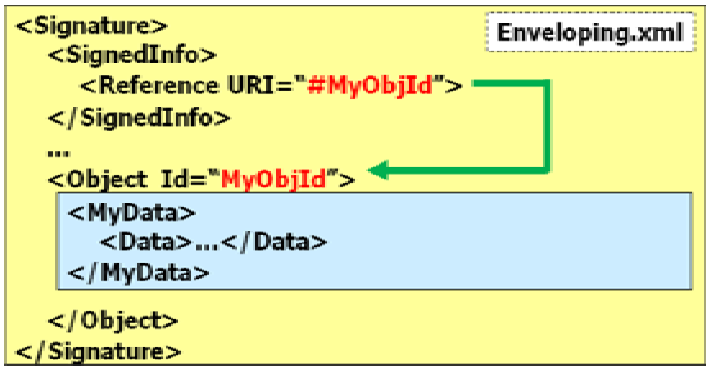
2.2.2. 同一ファイル内 XML 要素への Detached（外包）



2.2.3. 同一ファイルへの Enveloped（埋め込み）



2.2.4. XML 署名内オブジェクトへの Enveloping（内包）



2.3. 使用する署名ツール（署名ライブラリー）

XML 署名の機能は、Java 環境では Java6 より標準で JSR-105 の XMLSignature クラスが利用可能になった。Windows 環境であれば.NET Framework の SignedXml [8] クラスが標準で提供されている。XML 署名は手軽に利用できるようになったと言える。

2.3.1. Java6

Java 環境もこれまで標準では XML 署名の機能は無かったが、Java6 より標準で JSR-105 の XMLSignature クラスが提供されるようになった。Sun Java6 と IBM Java6 では同じプログラム（同一の class ファイル）が実行できた。まず Java6 の XMLSignature を用いた XML 署名（Enveloping、鍵生成）のコード例をリスト 1 に示す。

```
// ------------------------------------------------------------------------
// XML 形式の Enveloping の試験（RSA 鍵生成/KeyInfo も署名対象として Reference する）
private static void Enveloping() throws Exception
{
    String saveFile = "Enveloping-java.xml";
    System.out.println("----- Enveloping Test -----");

    // 署名ドキュメントの準備
    DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
    dbf.setNamespaceAware(true);
    Document document = dbf.newDocumentBuilder().newDocument();

    // 署名対象データの準備
    ByteArrayInputStream inStream = new ByteArrayInputStream(testXml.getBytes());
    Document refDoc = dbf.newDocumentBuilder().parse(inStream);
    Element element = refDoc.getDocumentElement();

    // XMLSignatureFactory の DOM 実装を取得する
    XMLSignatureFactory xsf = XMLSignatureFactory.getInstance("DOM");

    // 署名対象を Object として生成する
    XMLStructure content = new DOMStructure(element);
    XMLObject obj =
        xsf.newXMLObject(Collections.singletonList(content), "MyObjectId", null, null);

    // Reference を生成する
    DigestMethod dm = xsf.newDigestMethod(DigestMethod.SHA1, null);        // SHA1
    Reference reference = xsf.newReference("#MyObjectId", dm, null, null, null);

    // 2048bit の RSA 鍵を生成してセット
    KeyPairGenerator kpg = KeyPairGenerator.getInstance("RSA");
    kpg.initialize(2048);
    KeyPair key = kpg.generateKeyPair();

    // KeyValue 生成と KeyInfo を作成してセット
    KeyInfoFactory kif = xsf.getKeyInfoFactory();
    KeyValue keyValue = kif.newKeyValue(key.getPublic());
    KeyInfo keyInfo = kif.newKeyInfo(Collections.singletonList(keyValue), "MyKeyInfoId");

    // SignedInfo を生成する
    CanonicalizationMethod cm =
        xsf.newCanonicalizationMethod(CanonicalizationMethod.INCLUSIVE,
        (C14NMethodParameterSpec)null);
    SignatureMethod sm =
```

```
        xsf.newSignatureMethod(SignatureMethod.RSA_SHA1,null);
    List<Reference> references = new ArrayList<Reference>();
    references.add(reference);
    SignedInfo signedInfo = xsf.newSignedInfo(cm, sm, references);

    // Signature を生成する
    XMLSignature signature =
        xsf.newXMLSignature(signedInfo, keyInfo, Collections.singletonList(obj),
       "Enveloping", null);

    // DOM 用署名情報をセット
    DOMSignContext dsc = new DOMSignContext(key.getPrivate(), document);

    // 署名
    signature.sign(dsc);

    // XML 署名文書を出力。
    OutputStream os = new FileOutputStream(saveFile);
    TransformerFactory tf = TransformerFactory.newInstance();
    Transformer trans = tf.newTransformer();
    trans.transform(new DOMSource(document), new StreamResult(os));
}
```

リスト 1. Java6 による Enveloping 署名実装のサンプル

ここでは署名に利用する鍵は都度生成している。PKI を利用する場合は PKCS#12 形式の証明書＋秘密鍵を利用するか、Windows 証明書ストアに格納されている証明書＋秘密鍵を利用する事になる。PKI 的秘密鍵の利用方法に関しては 3.5 節で報告している。

Java6 を使って 4 種類の署名方式に対応する場合に注意が必要な点として、外部ファイルへの Detached で URI を間接指定にて署名･検証する際、現在のフォルダ位置をベース URI としてセットする必要があるので注意が必要となる。

今回は Sun Java6 の追検証を行う為に、IBM Java6 でも検証のみ行った。現在 IBM Java6 の Windows 版は提供されていないので、Linux のディストリビューションの 1 つである CentOS 3.5 の上で検証を行った。プログラミングの内容は Sun Java6 と全く同一であり、今回は同じ class ファイルを利用して検証を実行した。以下に Java6 の XMLSignature を用いた XML 署名の検証コード例（メイン部のみ）をリスト 2 に示す。

```
// ---------------------------------------------------------------------------
// 指定されたファイル中の XML 署名を全て検証
private static void Verify(String file) throws Exception
{
    DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
    dbf.setNamespaceAware(true);
    Document doc = dbf.newDocumentBuilder().parse(new FileInputStream(file));
```

```
    // Signature 要素を探す
    NodeList nl = doc.getElementsByTagNameNS(XMLSignature.XMLNS, "Signature");
    if (nl.getLength() == 0)
    {
        // Signature 要素が見つからない
        System.out.println("Verify: NG : cannot find Signature. : file = " + file);
        return;
    }

    // Document オブジェクトから、XMLSignature オブジェクトを生成
    XMLSignatureFactory fac = XMLSignatureFactory.getInstance("DOM");

    for(int i=0; i<nl.getLength(); i++)
    {
        // 検証対象と鍵取得クラスを取得
        Node target = nl.item(i);
        DOMValidateContext valContext =
            new DOMValidateContext(new KeyValueKeySelector(), target);

        // 表示用情報のセット
        String info = file;
        Node id = target.getAttributes().getNamedItem("Id");
        if(id != null)
            info += " : Id = " + id.getNodeValue();

        // XML から XMLSignature を非整列化する
        XMLSignature signature = fac.unmarshalXMLSignature(valContext);

        // 検証実行
        boolean coreValidity = signature.validate(valContext);

        if (coreValidity == false)
        {
            // 検証失敗
            System.out.println("Verify: NG : faild!!!!! : file = " + info);
            // 詳細検証
            boolean sv = signature.getSignatureValue().validate(valContext);
            System.out.println(" Signature validity status: " + sv);
            // Reference 先をチェック
            Iterator<Reference> it = signature.getSignedInfo().getReferences().iterator();
            for (int j = 0; it.hasNext(); j++)
            {
                boolean refValid = (it.next()).validate(valContext);
                System.out.println(" Reference[" + j + "] validity status: " + refValid);
            }
        }
        else
        {
            // 検証成功
            System.out.println("Verify: OK : valid. : file = " + info);
        }
    }
}
```

リスト 2. Java6 による検証実装のサンプル

2.3.2. .NET

.NET ではバージョン 2.0 より XML 署名の SignedXml クラスが提供されている。Windows 標準の API であるので、当然ではあるが Windows 証明書ストアの利用も容易である。ヘルプも豊富であり利用は比較的容易と感じた。.NET の SignedXml を用いた XML 署名（Enveloping、鍵生成）のコード例をリスト 3 に示す。

```
// ------------------------------------------------------------------------
// XML 形式の Enveloping の試験（RSA 鍵生成）
static void Enveloping()
{
    String saveFile = "Enveloping-dotn.xml";
    Console.WriteLine("----- Enveloping Test -----");

    // 署名対象データの準備
    XmlDocument document = new XmlDocument();
    document.PreserveWhitespace = true;     // 改行等を保存する（false は省略）
    document.LoadXml(testXml);

    // SignedXml の生成と署名アルゴリズムのセット
    SignedXml signedXml = new SignedXml();
    signedXml.Signature.Id = "Enveloping";
    signedXml.SignedInfo.SignatureMethod = SignedXml.XmlDsigRSASHA1Url; // RSA-SHA1

    // 署名対象を Object として追加する
    DataObject dataObject = new DataObject();
    dataObject.Data = document.ChildNodes;
    dataObject.Id = "MyObjectId";
    signedXml.AddObject(dataObject);

    // Reference を追加する
    Reference reference = new Reference();
    reference.Uri = "#MyObjectId";
    reference.DigestMethod = SignedXml.XmlDsigSHA1Url;  // SHA1
    signedXml.AddReference(reference);

    // 2048bit の RSA 鍵を生成してセット
    RSACryptoServiceProvider key = new RSACryptoServiceProvider(2048);
    signedXml.SigningKey = key;

    // KeyInfo を追加する
    KeyInfo keyInfo = new KeyInfo();
    keyInfo.Id = "MyKeyInfoId";
    keyInfo.AddClause(new RSAKeyValue(key));
    signedXml.KeyInfo = keyInfo;

    // 署名
    signedXml.ComputeSignature();

    // XML 署名の取得と保存
    XmlElement xmlSignature = signedXml.GetXml();
```

```
    XmlDocument saver = new XmlDocument();
    saver.PreserveWhitespace = true;
    saver.LoadXml(xmlSignature.OuterXml);
    saver.Save(saveFile);

    // 後始末
    key.Clear();
}
```

リスト 3. .NET による Enveloping 署名実装のサンプル

次に .NET の SignedXml を用いた XML 署名の検証コード例をリスト 4 に示す。

```
// ---------------------------------------------------------------------------
// 指定されたファイル中の XML 署名を全て検証
static void Verify(String file)
{
    // 検証対象データの準備
    XmlDocument document = new XmlDocument();
    document.PreserveWhitespace = true;     // 改行等を保存する（false にすると省略される）
    document.Load(file);

    // XML 署名の要素を探す
    XmlNodeList list =
        document.GetElementsByTagName("Signature", "http://www.w3.org/2000/09/xmldsig#");

    if (list == null || list.Count <= 0)
    {
        Console.WriteLine("Verify: NG : cannot find Signature. : file = " + file);
        return;
    }
    // 全ての XML 署名を順次検証
    foreach (XmlNode node in list)
    {
        // １署名の検証
        SignedXml verifier = new SignedXml(document);
        verifier.LoadXml((XmlElement)node);
        String info = file;
        if (node.Attributes["Id"] != null)
            info += " : Id = " + node.Attributes["Id"].Value;
        bool valid = verifier.CheckSignature();
        if (valid)
            Console.WriteLine("Verify: OK : valid. : file = " + info);
        else
            Console.WriteLine("Verify: NG : faild!!!!! : file = " + info);
    }
}
```

リスト 4. .NET による検証実装のサンプル

2.3.3. XML Security Library（XMLSec）

今回 Java6 と.NET 以外の XML 署名ライブラリーとして、Enveloping の検証を行う目的で XML Security Library (XMLSec) の Online Verifyier を利用した。URL と図 2. に画面サンプルを示す。

URL http://www.aleksey.com/xmlsec/xmldsig-verifier.html

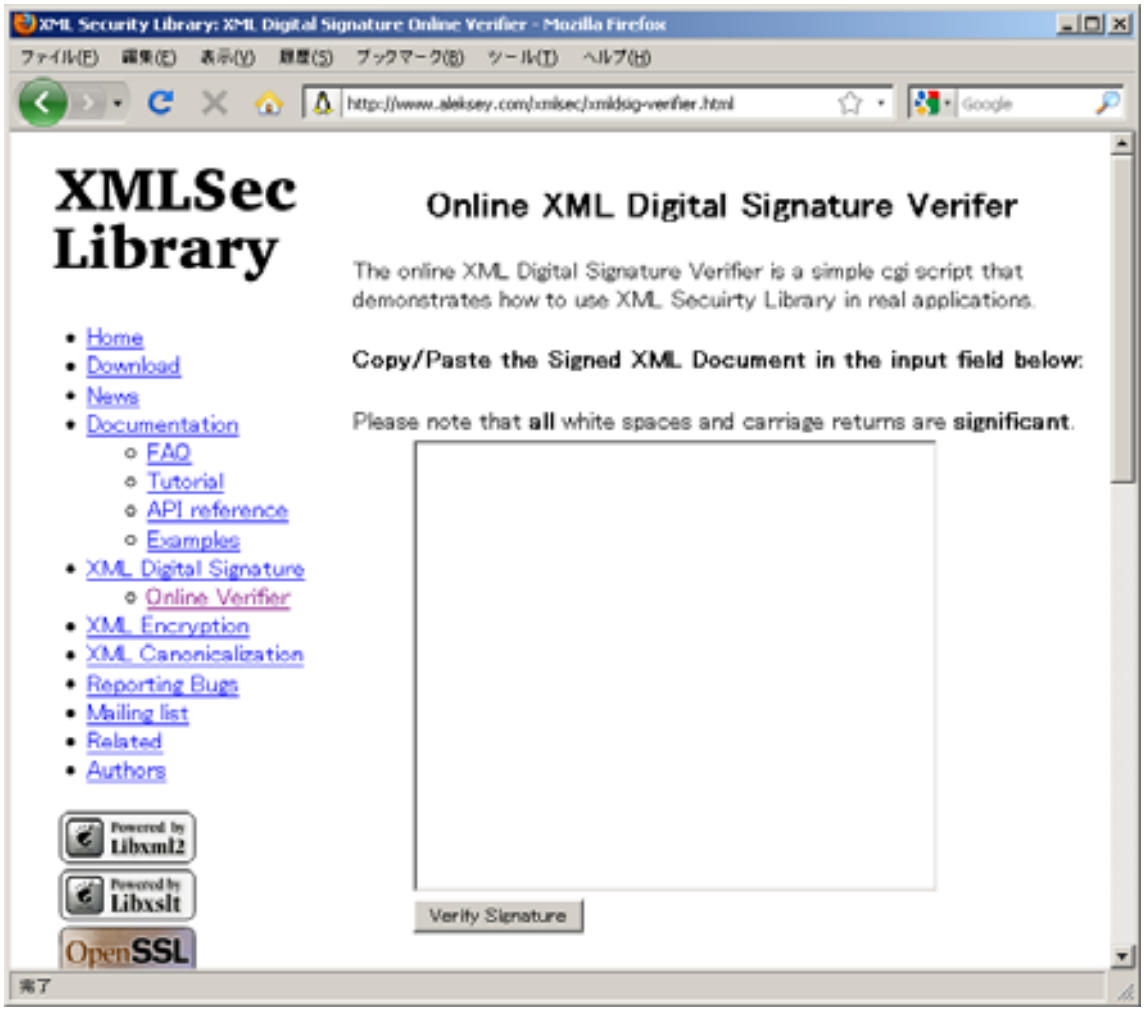


図 2. XMLSec の Online Verifier 画面

この画面を開き、検証対象の XML をコピー＆ペーストして Verify Signature ボタンで検証を行った。なお XMLSec は Linux でも提供されておりコマンドラインでも検証は出来るはずではあるが、今回は簡易に Web 経由での検証のみとした。

2.4. 検証手順

2.4.1. 署名対象の用意

今回は以下の XML データを署名対象とした。同一ファイル内署名の場合には XML 要素として、外部ファイルへの署名の場合には XML ファイル（target.xml）として用意した。

```
<MyData><Data Id="D1">book</Data><Data Id="D2">note</Data></MyData>
```

2.4.2. Sun Java6 による署名と検証

以下の手順で用意した署名対象に対し Sun Java6 署名ツールで XML 署名ファイル（4 個）を作成して検証する。

(1) 署名：Sun Java6 署名ツールで署名(Sign)して、XML 署名ファイル(4 個)を作成

(2) 検証：(1)の 4 ファイルを Sun Java6 署名ツールで検証(Verify)して結果を記録

2.4.3. .NET による署名と検証

以下の手順で用意した署名対象に対し.NET 署名ツールで XML 署名ファイル（4 個）を作成して検証する。

(3) 署名：.NET(C#)署名ツールで署名(Sign)して、XML 署名ファイル(4 個)を作成

(4) 検証：(1)の 4 ファイルを.NET 署名ツールで検証(Verify)して結果を記録

2.4.4. Sun Java6 による署名を.NET で検証

2.4.2 節の Sun Java6 署名ツールで作成された XML 署名ファイル（4 個）を.NET 署名ツールで検証する。

(5) 検証：2.4.2 で作成した 4 ファイルを.NET 署名ツールで検証(Verify)して結果記録

2.4.5. .NET による署名を Sun Java6 で検証

2.4.3 節の.NET 署名ツールで作成された XML 署名ファイル（4 個）を Sun Java6 署名ツールで検証する。

(6) 検証：2.4.3 で作成した 4 ファイルを Sun Java6 署名ツールで検証(Verify)して結果を記録

2.4.6. Sun Java6 による署名を IBM Java6 で検証

2.4.2 節の Sun Java6 署名ツールで作成された XML 署名ファイル（Enveloping）を IBM Java6 署名ツール検証する。

(7) 検証：2.4.2 で作成した 4 ファイルのうち Enveloping を IBM Java6 署名ツールで検証(Verify)して結果を記録

2.4.7. .NET による署名を IBM Java6 で検証

2.4.3 節の.NET 署名ツールで作成された XML 署名ファイル（4 個）を IBM Java6 署名ツールで検証する。

(8) 検証：2.4.3 で作成した 4 ファイルのうち Enveloping を IBM Java6 署名ツールで検証(Verify)して結果を記録

2.4.8. Sun Java6 による署名を XMLSec で検証

2.4.2 節の Sun Java6 署名ツールで作成された XML 署名ファイル（Enveloping）を XMLSec のオンライン検証で検証する。

(9) 検証：2.4.2 で作成した 4 ファイルのうち Enveloping を XMLSec のオンラインで検証(Verify)して結果を記録

2.4.9. .NET による署名を XMLSec で検証

2.4.3 節の.NET 署名ツールで作成された XML 署名ファイル（Enveloping）を XMLSec のオンライン検証で検証する。

(10) 検証：2.4.3 で作成した 4 ファイルのうち Enveloping を XMLSec のオンライン検証ページで検証(Verify)して結果を記録

3. 検証結果

3.1. 結果サマリー

3.1.1. .NET 署名に対する検証結果

署名方式	.NET で検証	Sun Java6 で検証	IBM Java6 で検証	XMLSec で検証
外部 Detached	○	○	---	---
内部 Detached	○	○	---	---
Enveloped	○	○	---	---
Enveloping（名前空間名指定無し）	○	×	○	○
Enveloping（名前空間名指定あり）	○	○	○	○

ほとんどの方式において、異種ツールでも検証が成功した。 Enveloping の名前空間名無しでは、Sun Java6 でのみ検証に失敗している。このケースでは相互運用性が実現できていない部分があると判断できる。

3.1.2. Sun Java6 署名に対する検証結果

署名方式	.NET で検証	Sun Java6 で検証	IBM Java6 で検証	XMLSec で検証
外部 Detached	○	○	---	---
内部 Detached	○	○	---	---
Enveloped	○	○	---	---
Enveloping（名前空間名指定無し）	×	○	×	×
Enveloping（名前空間名指定あり）	○	○	○	○

ほとんどの方式において、異種ツールでも検証が成功した。 Enveloping の名前空間名無しでは、.NET と IBM Java6 と XMLSec にて検証に失敗している。このケースにおいては相互運用性が実現できていない部分があると判断できる。

3.1.3. 検証結果に関する考察と現時点における推奨

Enveloping の名前空間名無しのケースを除けば相互運用性は確保されていると言える。Enveloping の名前空間名無しの問題は、実運用の際に署名対象に対して名前空間名

を明示的に指定する事で回避できる。現段階では Enveloping の利用では名前空間名（xmlns）の指定が推奨される。Enveloping の名前空間無しで何が問題になっているかは次の 3.2.節にて考察して明らかにしている。

3.2. 名前空間名指定無し Enveloping の問題考察

Sun Java6 と.NET の間で Enveloping の名前空間無しの場合に互換性が無い問題に関しては解析結果から見て、Sun Java6 と他のツールでは処理が異なっていると思われる。今回作成したテスト･プログラムの実装方法が間違っている可能性は否定できないが、全く同じプログラムを利用して異なる検証結果になっており、プログラム実装方法の問題である可能性は少ないと思われる。また各ツール単体では署名と検証で結果が一致しているので、単体で利用しているとこの問題は気づかない可能性が高い。

前述のように検証としては一般的であり特殊な実装では無いと考えている。Java6 では公開鍵の取得等も自分で実装する必要があるが、今回の問題は実は署名値の計算部分では無い。Sun Java6 で Enveloping の名前空間名無しの.NET の結果を検証すると以下となる。

```
Verify: NG : faild!!!!! : file = Enveloping-dotn.xml : Id = Enveloping
Signature validity status: true
Reference[0] validity status: false
```

署名値（Signature validity status）は正常だが、Reference 先（この場合は Object 要素下のテスト XML 要素）のハッシュ値が異なっていると表示されている。つまり Enveloping の名前空間名無しに関する問題は Enveloping されているデータの正規化問題と言える。この辺りをもう少し詳しく分析する。まず Reference されている署名対象の Object 要素は以下となる。なお以下は見やすくする為に要素毎に改行を入れているので注意。

```
<Object Id="MyObjectId" xmlns="http://www.w3.org/2000/09/xmldsig#">
<MyData xmlns="">
<Data Id="D1">book</Data>
<Data Id="D2">note</Data>
</MyData>
</Object>
```

MyData 要素の名前空間名は xmlns="" と明示的に空文字指定されている。これは Sun Java6 と.NET のどちらで署名してもこのようになっている。Object 要素では xmldsig 標準の名前空間が使われている。

最初に.NET の結果を見てみる。.NET で署名した場合に Reference のハッシュ値は Base64 で以下の値となっている。

```
<DigestValue>/nnmodoGCrYGeNgg8NLNG+1TCns=</DigestValue>
```

このダイジェスト値と一致するデータは以下のようになる。これも改行と空白が加えてある。

```
<Object xmlns="http://www.w3.org/2000/09/xmldsig#" Id="MyObjectId">
<MyData xmlns="">
<Data Id="D1">book</Data>
<Data Id="D2">note</Data>
</MyData>
</Object>
```

次に Sun Java6 で作成した署名ファイルにおいて Reference のハッシュ値は Base64 で以下の値となっている。これは.NET のハッシュ値とは異なっている。

```
<DigestValue>jZErUK6lIAY7Wjl7rDcw8eVIVtg=</DigestValue>
```

このダイジェスト値と一致するデータは以下となる。これも改行と空白が加えてある。

```
<Object xmlns="http://www.w3.org/2000/09/xmldsig#" Id="MyObjectId">
<MyData>
<Data Id="D1">book</Data>
<Data Id="D2">note</Data>
</MyData>
</Object>
```

これは MyData 要素に指定してあった xmlns="" が省略されている。

なお Sun Java6 にて正規化のみを Object 要素以下に適用した場合には xmlns="" は省略されない。なぜ署名時にこのような正規化がされているかは不明である。以下に同じ Linux 環境（CentOS3.5）にて同じバイナリ（class ファイル）を利用して、Sun Java6 と IBM Java6 の両方で Sun Java6 と.NET で生成した名前空間名無し Enveloping 結果ファイルを検証した例を示す。なお Enveloping-java.xml は Sun Java6 で生成した名前空間名無しの Enveloping 署名ファイルであり、Enveloping-dotn.xml は.NET Framework で生成した名前空間名無しの Enveloping 署名ファイルとなっている。

```
// Sun Java6 における実行結果.
[centos53]$ java -version
java version "1.6.0_17"
Java(TM) SE Runtime Environment (build 1.6.0_17-b04)
```

```
Java HotSpot(TM) Client VM (build 14.3-b01, mixed mode, sharing)
[centos53]$ java -cp . XmlSignTest Enveloping-dotn.xml
----- START OF VERIFY -----
Verify: NG : faild!!!!! : file = Enveloping-dotn.xml : Id = Enveloping
 Signature validity status: true
 Reference[0] validity status: false
----- END OF VERIFY -----
[centos53]$ java -cp . XmlSignTest Enveloping-java.xml
----- START OF VERIFY -----
Verify: OK : valid. : file = Enveloping-java.xml : Id = Enveloping
----- END OF VERIFY -----
```

```
// IBM Java6 における実行結果.
[centos53]$ java -version
java version "1.6.0"
Java(TM) SE Runtime Environment (build pxi3260sr6-20090925_01(SR6))
IBM J9 VM (build 2.4, JRE 1.6.0 IBM J9 2.4 Linux x86-32 jvmxi3260sr6-
20090923_42924 (JIT enabled, AOT enabled)
J9VM - 20090923_042924
JIT  - r9_20090902_1330ifx1
GC   - 20090817_AA)
JCL  - 20090924_01
[centos53]$ java -cp . XmlSignTest Enveloping-dotn.xml
----- START OF VERIFY -----
Verify: OK : valid. : file = Enveloping-dotn.xml : Id = Enveloping
----- END OF VERIFY -----
[centos53]$ java -cp . XmlSignTest Enveloping-java.xml
----- START OF VERIFY -----
Verify: NG : faild!!!!! : file = Enveloping-java.xml : Id = Enveloping
 Signature validity status: true
 Reference[0] validity status: false
----- END OF VERIFY -----
```

4. 付録（その他確認）

4.1. その他確認事項

電子署名で使われる技術もアルゴリズム危殆化等に対応する為に更新されている。特に

SHA-1 ハッシュアルゴリズムに関しては、内閣官房情報セキュリティセンターより 2008 年 4 月に「政府機関の情報システムにおいて使用されている暗号アルゴリズム SHA-1 及び RSA1024 に係る移行指針」 [9] により、公的利用としては 2013 年度に SHA-2 への適合が求められている。SHA-2 対応に関しては 4.2 節で対応状況をまとめた。他にも XML 正規化（C14N）仕様も W3C より 1.1 [10] の Update が公開されているので 4.3 節に確認結果をまとめた。電子署名で良く使われる PKCS#12 ファイルや Windows 証明書ストアの利用が可能かどうかを 4.4 節にて確認結果をまとめた。最後に XML を使った電子署名関連では良く使われる Base64 変換他に関して 4.5 節にて確認結果をまとめた。

4.2. SHA-2（SHA-256/384/512）への対応状況の確認

Reference 要素のダイジェスト方式として SHA512（SHA-2 / 512bit）を指定してみた。SHA512 は .NET Framework と Java6 の両方で対応していた。SHA-2 シリーズの他のビット数である SHA384 と SHA256 も利用可能であった。

```
// Java6 においてダイジェスト方式として SHA-512 を指定
DigestMethod dm = xsf.newDigestMethod(DigestMethod.SHA512, null); // SHA2
Reference reference = xsf.newReference("#MyObjectId", dm, null, null, null);
```

```
// .NET においてダイジェスト方式として SHA-512 を指定
Reference reference = new Reference();
reference.Uri = "#MyObjectId";
reference.DigestMethod = "http://www.w3.org/2001/04/xmlenc#sha512"; // SHA2
signedXml.AddReference(reference);
```

リスト 5. Java6 と.NET にて Reference への SHA2 ダイジェスト方式の指定

次に XML 署名方式として RSA-SHA512 を指定してみた。これは残念ながら .NET Framework と Java6 の両方で未対応であった。システムとしてはブラウザ（IE 等）において RSA-SHA2 を利用したサーバ証明書も利用可能である状況だが、API である XML 署名では調査時点では未対応であった。早期対応を望みたい。なお RSA 暗号の鍵長として 2048 ビットへは両方で対応済みであった。

```
// Java6 において署名方式として RSA-SHA512 を指定
SignatureMethod sm =
    xsf.newSignatureMethod("http://www.w3.org/2001/04/xmldsig-more#rsa-sha512",null);
// 署名時に例外（NoSuchAlgorithmException: unsupported algorithm）が発生する
```

```
// .NET において署名方式として RSA-SHA512 を指定
SignedXml signedXml = new SignedXml();
signedXml.SignedInfo.SignatureMethod =
    "http://www.w3.org/2001/04/xmldsig-more#rsa-sha512";
// 署名時に以下の例外
// (CryptographicException =
//     "指定された署名アルゴリズムの SignatureDescription を作成できませんでした。)
```

リスト 6. Java6 と.NET にて署名方式として RSA-SHA2 ダイジェスト方式の指定

4.3. XML 正規化（C14N）の確認

XML 正規化（C14N 1.0）の単独利用が可能かどうかを調査したが .NET Framework と Java6 の両方で利用可能だった。しかし W3C の Update である最新の C14N 1.1 には調査時点ではどちらも未対応であった。

```
// Java6 において C14N 変換
XMLSignatureFactory xsf = XMLSignatureFactory.getInstance("DOM");
CanonicalizationMethod cm =
    xsf.newCanonicalizationMethod(CanonicalizationMethod.INCLUSIVE,
    (C14NMethodParameterSpec)null);
OctetStreamData rsltData =
    (OctetStreamData)cm.transform(new OctetStreamData(inStream), null);
```

```
// .NET において C14N 変換
Transform trans = new XmlDsigC14NTransform();
trans.LoadInput(doc);
MemoryStream st = (MemoryStream)trans.GetOutput();
st.Position = 0;    // 先頭に戻してから利用
```

リスト 7. **Java6 と.NET にて C14N による正規化のみ使う**

4.4. PKCS#12 ファイルや Windows 証明書ストア利用方法の確認

PKI として電子署名を使う場合には X.509 証明書と秘密鍵をペアにして利用する利用方法が一般的である。今回は Java6 も Windows 環境にてテストを行ったので Windows 証明書ストアの利用も試してみた。結果としては .NET Framework と Java6 の両方において、PKCS#12 形式の証明書＋秘密鍵も Windows 証明書ストアも利用できた。

```
// Java6 において PKCS#12 ファイルから読み込み
KeyStore ks = KeyStore.getInstance("PKCS12");
FileInputStream fis = new FileInputStream(pkcs12File);
ks.load(fis, pkcs12Pswd.toCharArray());
for (Enumeration<String> e = ks.aliases(); e.hasMoreElements() ;)
{
    String alias = e.nextElement();
    if(ks.getKey(alias, pkcs12Pswd.toCharArray()) != null)
    {
        // PKCS#12 ファイルに含まれ最初に秘密鍵を持つ証明書の alias 名
        myAlias = alias;
        break;
    }
}
// 証明書と公開鍵と秘密鍵の取得
Certificate cert = ks.getCertificate(myAlias);
PublicKey pubKey = cert.getPublicKey();
PrivateKey privKey = (PrivateKey)ks.getKey(myAlias, pkcs12Pswd.toCharArray());
```

```
// .NET において PKCS#12 ファイルから読み込み
X509Certificate2 cert = new X509Certificate2(pkcs12File, pkcs12Pswd);
RSACryptoServiceProvider key = (RSACryptoServiceProvider)cert.PrivateKey;
```

リスト 8. Java6 と.NET にて PKCS#12 ファイルを利用

```
// Java6 において Windows 証明書ストアから取得
KeyStore ks = KeyStore.getInstance("WINDOWS-MY");
ks.load(null, null);
myAlias = "Week2009";     // xmlc_cert.p12 の alias 名
// 証明書と公開鍵と秘密鍵の取得
Certificate cert = ks.getCertificate(myAlias);
PublicKey pubKey = cert.getPublicKey();
PrivateKey privKey = (PrivateKey)ks.getKey(myAlias, pkcs12Pswd.toCharArray());
```

```
// .NET において Windows 証明書ストアから取得
X509Store store = new X509Store("MY", StoreLocation.CurrentUser);
store.Open(OpenFlags.ReadOnly | OpenFlags.OpenExistingOnly);
// 個人ストアから証明書を抽出
X509Certificate2Collection collection = (X509Certificate2Collection)store.Certificates;
// 有効な証明書を更に抽出
X509Certificate2Collection fcollection =
    (X509Certificate2Collection)collection.Find(X509FindType.FindByTimeValid,
     DateTime.Now, false);
// ダイアログで選択
X509Certificate2Collection scollection =
    X509Certificate2UI.SelectFromCollection(fcollection, "Test Certificate Select",
    "Select a certificate from the following list to get information on that certificate",
    X509SelectionFlag.SingleSelection);
// 証明書取得：キャンセルされると scollection.Count = 0 になるがチェックは省いている
X509Certificate2 cert = scollection[0];
```

リスト 9. Java6 と.NET にて Windows 証明書ストアを利用

4.5. その他（Base64 機能等）

XML 署名を使う場合にバイナリの情報は一般的に Base64 にてエンコードされている。この為に Base64 のエンコードとデコードの機能が提供されていると XML 署名の利用が容易になる。Base64 に関して .NET Framework では Convert::ToBase64String() と Convert::FromBase64String() が提供されている。しかし Java6 の標準では Base64 の機能は提供されていない。Java 環境でも Base64 に関するクラスは比較的容易に入手できるので大きな問題では無いが、出来れば標準にて提供されることを今後期待したい。

他に同一ファイル内での複数署名や、複数 Reference（署名対象）に関しては .NET Framework も Java6 も問題無く利用ができた。Reference 先に KeyInfo 要素を指定するケースに関して .NET では単純にはうまく使えなかったが、今回は未調査とした。

5. まとめ（終わりに）

今回の検証結果としては、名前空間名指定無しの Enveloping においてツール間で互換性が無い部分が見つかった。電子署名で利用する PKI とはパブリックな共通基盤であるのでこのような問題が本当に存在しているのであれば解決すべき問題と言える。今後関係ツールの開発組織に問題としてインプットして行きたいと考えている。

Java6 も .NET Framework も、XML 署名の機能が標準提供されるようになった事は、容易に利用が出来るようになった点において非常に大きい意味があると考えられる。しかしながら SHA-2 や正規化（C14N）も最新の 1.1 への対応と言う面では対応が遅いように思われる。この辺りは小回りの効くオープンソース系の実装の方が有利と言えるのかもしれない。目的や将来の運用も考えてどのような実装を利用するか検討する必要があるだろう。

電子署名が使われる PKI の世界では相互運用性は非常に重要な課題である。本来は標準への適合性の宣言をライブラリー開発元からなされるべきであり、更には何らかの適合性確認の為の試験データやルールが公的または第三者機関により整備公開されるべきであろう。将来的にそのような取り組みが実現することを期待したい。今回の調査と本書がそのような取り組みのきっかけになればと願っている。

6. 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 - XML コンソーシアム
http://www.xmlconsortium.org/public_doc/mof2008_security/mof2008.html

[2] XML 暗号化･電子署名ツール検証報告 - 第 8 回 XML コンソーシアム Week
http://www.xmlconsortium.org/seminar09/090512-13+19-20/090512-prog.html

[3] Java SE Downloads - Sun Developer Network (SDN)
http://java.sun.com/javase/downloads/index.jsp

[4] developerWorks : Java technology : IBM developer kits : Linux Download information.
http://www.ibm.com/developerworks/java/jdk/linux/download.html

[5] XML Security Library (XMLSec Library)

http://www.aleksey.com/xmlsec/

[6] ECOM CAdES/XAdES Plugtest 2007 - 2007 年度 相互運用性テスト プロジェクト
http://www.ecom.jp/longtermstorage/interoptest2007.html

[7] XML Signature Syntax and Processing (Second Edition) - W3C
http://www.w3.org/TR/xmldsig-core/

[8] .NET Framework クラス ライブラリー SignedXml クラス
http://msdn.microsoft.com/ja-jp/library/system.security.cryptography.xml.signedxml%28VS.71%29.aspx

[9] 政府機関の情報システムにおいて使用されている暗号アルゴリズム SHA-1 及び RSA1024 に係る移行指針 - 内閣官房情報セキュリティセンター
http://www.nisc.go.jp/active/general/pdf/crypto_pl_draft.pdf

[10] Canonical XML Version 1.1 - W3C
http://www.w3.org/TR/xml-c14n11/