



UNIVERSITY OF WESTERN SYDNEY

College of Science, Technology and Environment
School of Computing and Information Technology
Centre for Advanced Systems Engineering

XML Security Using XSLT

Robert Bartlett

rbartlet@cit.uws.edu.au

Malcolm Cook

mal@cit.uws.edu.au

Technical Report No. CIT/15/2002

October 2002

University of Western Sydney
Locked Bag 1797, Penrith South DC, NSW 1797
Australia

XML Security Using XSLT

R. G. Bartlett and M. W. Cook
Centre for Advanced Systems Engineering
University of Western Sydney
rbartlet@cit.uws.edu.au, m.cook@uws.edu.au

Abstract

The eXtensible Markup Language (XML) is regarded generally as having promise of becoming established as the general purpose framework for enabling transfer of data amongst heterogeneous environments. It is of interest therefore to analyse how suitable it may be once details of applications requirements and constraints are taken into account. One important requirement is for the security of documents in transit.

Closely associated with XML is the eXtensible Stylesheet Language (XSL), whose document transformation component (XSLT) may well have sufficient functionality to perform all reasonable cryptographic transformations to deliver a desired level of document security. We examine this question by describing a real world XML application whose security requirements are more complex than for a simple document transfer between just two parties; proposing a document transfer architecture into which XSLT can be plugged-in; and identifying those features of XSLT which must be applied to meet the application requirements.

We conclude that XSLT is only just adequate in the proposed scenario; and then only by making use of its "extension functions" capability.

Keywords: XML, XSLT, Security.

1. Introduction

Over the past three years the eXtensible Markup Language (XML) has appeared, offering a framework which promotes the movement of business information across networks. XML is a standard from the World Wide Web Consortium (W3C), allowing users to structure (and label) information separately from the presentation of that information. A single instance of this structured information is known as an XML Document. An XML document must adhere to particular syntax and semantics as outlined in Extensible Markup Language (XML) 1.0

(Second Edition), W3C Recommendation 6th October 2000 [1].

The use of XML, as a container for business information is desirable because it is platform independent; it utilizes existing communication protocols (i.e. HTTP and SSL); and it is able to pass through firewalls as a result [4].

Through adherence to a system independent character set (Unicode), XML documents are human legible. While this makes an XML document more palatable for viewing purposes (compared to a binary file), it raises privacy of information issues. At the very least, information required to be secure should not appear human legible. A simple, yet effective way to achieve this is to encrypt XML data elements. For reasons to be explained below, encryption of elements should be carried out on an individual (isolated) basis. Within an XML document there are two different places data elements can reside:

- As attributes of tags, or
- As text children of content elements.

We will require a secure XML document to retain the same structure as the original XML document. By doing this, an application can be decoupled from any security measures. This principle is grounded in practical experience with XML applications such as the *Data Sockets* project [8], where documents are composed with elements from different sources, by an intermediary who ought not to be privy to the data content of those elements, as well as providing for a finely-grained access control mechanism in respect of the visibility of individual elements of a document. This rules out a simplistic approach, i.e. to encrypt the entire contents of an XML document (tags included) at source, producing unstructured text. Such text could be encapsulated in an XML content element, hence making the document an "XML document" (providing encoding types and character sets are supported).

Miyazawa and Kushidia [7] present a similar approach to XML document security. In their scheme

security information, including the name of the algorithm and size of the key used for encryption is included inside of a “secure XML document”. Whilst this is not an explicit vulnerability, keeping this information separate from the XML document, makes a secure XML document more difficult (but not impossible) to decrypt; as information pertaining to encryption specifics is commonly used as the foundation for an unauthorized party to decrypt documents.

XSLT (eXtensible Stylesheet Language Transformations) is a W3C specification for a document manipulation language capable of restructuring documents and performing computations on their elements. It is of interest to discover whether XSLT could express transformations of an XML document to and from an encrypted form, thereby unifying this aspect of document handling within the XSLT framework. This leads to the questions addressed in this paper: What are the issues involved with using XSLT to transform an XML document into a secured format and back again; and how suitable is this use of XSLT?

If we regard encryption/decryption as just another XML document transformation operation, then it is apparent that the advantages XSLT provides in respect of other operations, such as presentation formatting, carry over to this aspect of document transfer as well. In particular it might avoid the need to deploy and maintain special purpose software: Once a standard XSLT processor is available, the specifics of security operations may be confined to stylesheets. As we shall see however, it turns out that XSLT can't provide a complete solution.

As mentioned above, in this paper we consider an e-business scenario where an intermediary or broker parses input XML documents and produces a collection of (smaller) XML documents (comprising a transaction), or correspondingly, where this intermediary combines a collection of XML documents. Data contained in the XML documents should be hidden from the intermediary (to respect the privacy of information), but the structure of a document, including identification of individual elements is to remain visible to enable the broker to function. It ought to be possible to apply encryption/decryption stylesheets in this scenario, so that information can be hidden as desired from the intermediary, and at the same time leave control of information to its source: A result of applying encryption/decryption stylesheets is that provision for an access control mechanism becomes possible; by varying either encryption key, or even encryption procedure, element by element.

In what follows we examine the capability of XSLT to perform security transformations meeting the above requirements and identify its limitations in this regard. Although we reason directly from the W3C XSLT

specification to reach our conclusions, we also provide a brief description of the test environment used during the course of the project to verify those conclusions in the case of a particular XSLT processor implementation.

2. Related Work

Although this research set out to ascertain the suitability and applicability of using XSLT to achieve XML document security, it is not the only way in which XML document security can be achieved. A simple yet effective way, adhering to the basis of element-wise encryption and utilizing an XML processor, can be constructed in almost any programming language. The obvious benefit of such an approach is a reduction in the amount of complexity required to construct and implement. Problems with this approach are that such a package is required at each point in the system (where the security of the XML document is dealt with) and a new mechanism for determining selection for encryption within a source XML document is required.

An example of an implementation of this type is that of W3C's, XML Encryption, which incorporates security concerns into an XML processor, controlled by attributes in an XML document (for the purposes of initialization and invocation) [6]. A realization of this processor is the XML Security Suite from IBM [3].

2.1. Element-Wise Encryption

Element-Wise Encryption is the name given to the process of applying encryption algorithms to sections of an XML document to produce a secure XML document. This concept was first proposed by Maruyama and Imamura [5], in an article published on a W3C mailing list. Unlike the definition contained in this research Maruyama and Imamura required stricter validation of information and allowed resulting documents to possess different structures (to the original document). Applying element-wise encryption can be likened to carrying out semantic operations on sections of an XML document.

In Maruyama and Imamura's proposal of element-wise encryption, validation of information (through the use of a Document Type Definition [DTD]) was suggested, as means of assuring information contained within an XML document existed within an appropriate form. This article does not consider issues derived or relating to the validation of XML documents.

3. Encryption Transformations

In this section we examine how readily “pure” XSLT may be applied to encryption transformations, and

conclude that the only reasonable strategy is to employ *extension functions*, which allow XSLT code to request computations expressed in another language. The proposed model for this problem is depicted in Figure 1: A source XML document undergoes a transformation (contained in a stylesheet), to produce an encrypted XML document (that is, a secure XML document); a second stylesheet transforms an encrypted XML document into its original form.

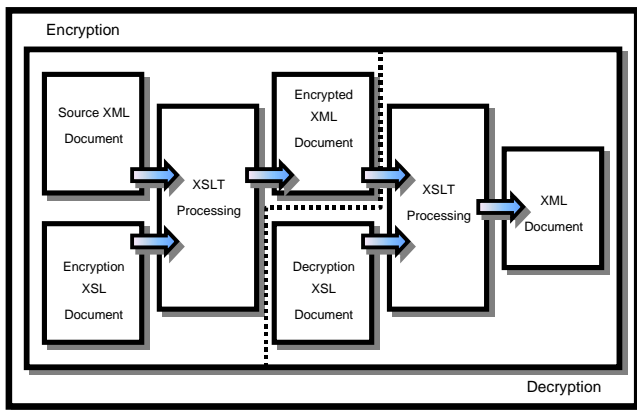


Figure 1. XML Encryption and Decryption

Constructing encryption and decryption stylesheets with complete adherence to XSLT syntax is plausible, but not fruitful owing to the inherent complexity of programming encryption algorithms without an assignment statement. Even simple programming within a stylesheet significantly increases complexity - one need only consider the effort needed to compute $N!$ (the factorial function) using an XSL stylesheet to be convinced of this.

3.1. Extension Functions

In order to provide conventional programming language functionality without adversely affecting stylesheet complexity, extension function provisions (Section 14, in [2]) are utilized. Requirements for two different extension functions are immediately obvious:

1. A function to act as an encryption interface between cryptographic libraries and a stylesheet.
2. A function to act as a decryption interface between cryptographic libraries and a stylesheet.

These extension functions are to be applied in accordance with element-wise encryption. A stylesheet carrying out the encryption or decryption of XML

documents is conceptualized as consisting of several components:

- Template declarations matching all tags in a corresponding XML document. This must be done to preserve the structure contained within the original XML document. A demonstration of this is an XML document:


```
<?xml version="1.0" ?>
<tag1>
  <tag2/>
</tag1>
```

 containing two different tags; tag1 and tag2, must have each tag “matched” by a template, in order to be reproduced in an output document. An XSLT processor discards tags not explicitly “matched”. This is most undesirable.
- Extension functions, which carry out both encryption and decryption of text strings.
- Attribute identification and reproduction. This point is similar to the first, in that attributes contained in a source XML document must also be reproduced in an output document, else be discarded (and their information lost).

Inclusion of extension functions within the stylesheet framework is depicted in Figure 2, where an XSLT Processor accepts an XSLT document (a stylesheet) and an XML document, utilizes functionality from extension functions in order to produce either an encrypted or decrypted XML document.

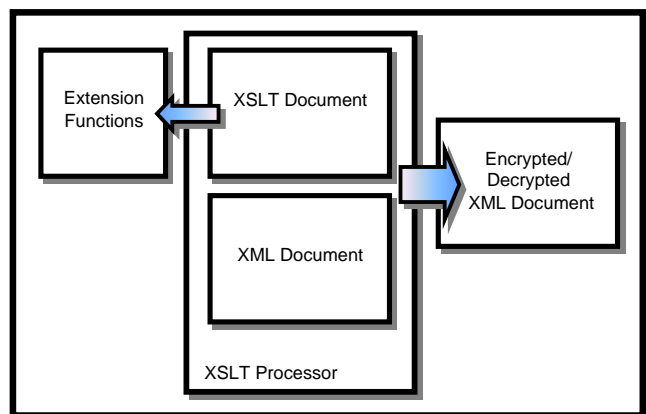


Figure 2. Extension Functions within the Stylesheet Framework

Clearly the underlying logic of this XSLT solution would be similar to that of a custom-built encryption suite as described in Section 2; i.e. parse then encrypt/decrypt. While this diminishes the advantages which might have

been gained from a “pure” XSLT solution, the pre-implemented logic within an XSLT processor may still be exploited to connect, in a reasonably transparent way, the encryption functions with an XML parser interface such as SAX [11]. We may thereby continue to expect to save some degree of development and deployment effort as pointed out in Section 2.

4. Element-Wise Encryption

In this section we discuss how the requirement for structure preserving encryption may be met, and examine what constraints are placed upon such transformations by the way in which XSLT processors operate.

If we require an encrypted XML document to have the same XML structure as its decrypted form, it can thereby be manipulated, irrespective of whether it is encrypted or not, transparently by document handling applications. Values contained in such a document are decoupled from the document structure. For instance, in a scenario where an XML document is received and forwarded, based upon the value of a specific document element, all values (except for the identifying element) are irrelevant (to the task of forwarding). The difference between encrypted and unencrypted documents is that in an encrypted document, the desired textual pieces of data are indecipherable. These pieces of data can either be tag attributes or text-node children of a content element. For example:

```
<tag attribute="data"> data </tag>
```

where `data` represents data able to be encrypted. `data` resides in two different locations, as a child of the `tag` element and the value of the `attribute` attribute.

Small amounts of encrypted data are commonly regarded as easier for an eavesdropper to decrypt, than a large encrypted block of data. Within this study this issue was disregarded, since many security suites compensate for the risk of encrypting small amounts of data (i.e. by padding data to a certain length before encrypting).

The type of encryption applied to any piece of data can be independent from the encryption applied to any other piece of data. For example, each attribute in an XML document may have a different encryption algorithm and key applied to it. Furthermore, pieces of data that can be encrypted do not necessarily have to be; depending on the requirements at the source of the information.

During the course of this project the concept of element-wise encryption needed refinement. Initially, element-wise encryption was envisaged to encrypt a segment of XML (a node and all of its children). This uncovered a problem: Before a stylesheet can be applied

to an XML document, an XML processor must parse that XML document, producing a DOM (Document Object Model) Object representation. The DOM is an API specification for operations on a document in parsed (deserialized) form [9]. During the application of a stylesheet, an XSLT processor steps through a DOM Object applying corresponding templates as tags are matched (tags in the XML document are “matched” with templates in the stylesheet). This means that before a stylesheet is applied, an XML document no longer exists (as an XML document). Whilst it was possible to pass a section of a DOM Object to the encryption function, this presented several new problems: a segment (sub-tree) of a DOM Object is not a text element; and when trying to restore a fragment, additional processing could not be carried out (in order to return a sub-tree into its original XML format).

In preliminary experiments, DOM segments were extracted from an XML document, by a stylesheet and encrypted. The encrypted DOM segments were immediately decryptable. However, when trying to serialize encrypted DOM objects (to be stored in an encrypted XML document) information about the objects is lost and attempts to restore DOM segments will be unsuccessful.

Without using additional extension functions no means were discovered to perform additional evaluation (i.e. value extraction) upon such DOM segments.

Owing to the implied additional level of complexity, it was concluded that element-wise encryption should not apply to “structured” elements, i.e. non-trivial subtrees of the document, and would be confined to text nodes.

5. Project Environment

Aspects of XSLT discussed in this paper were investigated and/or verified by means of specifying and implementing a number of demonstration stylesheets. The XSLT processor utilized in these tests was `jd.xslt` version 1.2.1 released on September 18th, 2001 by Johannes Döbler. `jd.xslt` claims conformance to the XSLT 1.1 Working Draft [2] and is constructed in the Java Programming Language. At present, `jd.xslt` is the only publicly available XSLT 1.1 compliant processor. With the discontinuation of the XSLT 1.1 Working Draft, additional implementations are not expected, somewhat limiting further testing.

As described in Section 3.1, extension functions are code written in a (programming) language other than XSLT that are “hooked” into a stylesheet to provide additional functionality. XSLT extension functions are processor implementation specific. XSLT 1.1 introduced the `<xsl:script>` tag for the declaration of extension

functions in an attempt to standardize their use (although this has yet to be resolved). In order for an XSLT processor to invoke an extension function written in a particular language, the processor must possess a binding to the language in which the extension function is written. Language bindings are optional and remain processor specific.

Extension functions used in the test scenario are written in Java. The reason for this is that `jd.xslt`, (the XSLT Processor) is constructed in Java and provides a binding for this language. In this scenario, the Java extension functions use the same Java Runtime Environment (JRE) as `jd.xslt` since `jd.xslt` requires a JRE in order to operate.

6. Test Procedures

This section gives an overview the series of steps used to carry out testing.

The file containing the Java extension function code must first be compiled using the “`javac`” compiler included with a Java Software Development Kit (SDK).

The source XML document and encryption stylesheet are passed as input to `jd.xslt`. The class file, created from the extension function source code once compiled, must be included on the classpath of the processor. By applying the encryption stylesheet to the source XML document, a secure XML document is produced, along with the temporary “keystore” file, containing a queue of the keys used for encryption.

The secure XML document and the decryption stylesheet are passed as input to `jd.xslt`. Again, the class file containing the extension functions must be included on the XSLT processor’s classpath. From this transformation, the source XML document is reproduced.

```
<?xml version="1.0"?>
<document>
  <customer id="1">
    <surName>Bloggs</surName>
    <firstName>Joe</firstName>
  </customer>

  <customer id="2">
    <surName>Doe</surName>
    <firstName>John</firstName>
  </customer>
</document>
```

Figure 3. Source XML Document

The encryption and decryption stylesheets used in testing are similar. Differences between the two reside in the different extension functions used (i.e. the encryption stylesheet uses the `encrypt` function, whilst the decryption stylesheet uses the `decrypt` function).

Figure 3 depicts a portion of an XML document which we’ll use to illustrate a typical test case; the

```
<xsl:script implements-prefix="security"
  language="java" src="java:test"/>
...
<xsl:template match="surName">
  <xsl:text
    disable-output-escaping="yes"
  >&lt;surName&gt;</xsl:text>
  <xsl:value-of select="security:encrypt(text())"/>
  <xsl:text disable-output-escaping="yes">&lt;surName&gt;
  </xsl:text>
</xsl:template>
```

Figure 4. Stylesheet Encryption Extract

element `surName` is to be encrypted. The required transformation appears in the corresponding stylesheet segment depicted in Figure 4. The encryption function `encrypt()` is identified as an extension function by means of the namespace prefix `security` which is itself defined and linked to the relevant compiled class by the `xsl:script` element.

7. Conclusions

While XSLT possesses the ability to manipulate the structure and contents of XML documents, trying to extract additional functionality, such as the application of an encryption algorithm leads to unacceptable complexity and constrains the ways in which documents can be encrypted.

It has been shown that through the use of extension functions the former problem can be overcome, by providing small modules that extend XSLT capabilities. This solution led to some technical difficulties, since, at the time of this project, extension functions had yet to be clearly (and cleanly) defined within an XSLT specification. Subsequent specification updates [10], where, incidentally, the `<xsl:script>` element has been removed, indicate that this is still the case.

It may be considered that, as a technology, XSLT is still immature, with the specifications classified as a “Working Draft” after four years of development. XML Encryption has progressed to the status of “Candidate Recommendation” after two years, and further development may see this established as the pathway of choice to XML document security.

8. References

- [1] W3C, "Extensible Markup Language (XML) 1.0 (Second Edition); W3C Recommendation 6 October 2000", <http://www.w3.org/TR/2000/REC-xml-20001006>, October 2000.
- [2] W3C, "XSL Transformations (XSLT) Version 1.1; W3C Working Draft 24 August 2001", <http://www.w3.org/TR/2001/WD-xslt-20010824/>, August 2001.
- [3] Tidwell D., "The XML Security Suite: Increasing the security of e-business", <http://www-4.ibm.com/software/developer/library/xmlsecuritysuite/index.html>, April 2000.
- [4] Nehren D., "XML Through the Wall", <http://www.xmlmag.com/upload/free/features/xml/2001/05may01/dn0102/dn0102.asp>, May 2001.
- [5] Maruyama H. and Imamura T., "Element-Wise XML Encryption", <http://lists.w3.org/Archives/Public/xml-encryption/2000Apr/att-0005/01-xmlenc.html>, April 2000.
- [6] W3C, "XML Encryption Syntax and Processing; WG Working Draft 26 June 2001", <http://www.w3.org/TR/2001/WD-xmlenc-core-20010626/>, June 2001.
- [7] Miyazawa T. and Kushida T., "An Advanced Internet XML/EDI Model Based on Secure XML Documents", *Parallel and Distributed Systems*, IEEE, Iwate, Japan, June 2000, pp. 295-300.
- [8] Bryan G., Curry J., McGregor C., Holdsworth D. and, Sharply R., "Using XML to facilitate information management across multiple local government agencies", *Hawaii International Conference on System Sciences*, Hawaii, United States, January 2002, pp. 1544-1553.
- [9] W3C, "Document Object Model (DOM) Level 3 Core Specification Version 1.0; W3C Working Draft 05 June 2001", <http://www.w3.org/TR/2001/WD-DOM-Level-3-Core-20010605>, June 2001.
- [10] W3C, "XSL Transformations (XSLT) Version 2.0; W3C Working Draft 30 April 2002", <http://www.w3.org/TR/2002/WD-xslt20-20020430/>, April 2002.
- [11] Brownell D., "SAX", <http://saxproject.org>, January 2002.