Design and Implementation an Efficient and Automatic Environment for Securing XML Documents

Gwan-Hwan Hwang and Tao-Ku Chang
Dept. of Information Management
National Chi-nan University
Puli, Taiwan
ghhwang@im.ncnu.edu.tw, s9213517@ncnu.edu.tw

Abstract

In this paper, we propose an infrastructure to support an efficient and automatic environment for securing XML documents. Based on a generalized encryption model of XML, the element-wise encryption, we define the Document Security Language (DSL) which is a language for specifying the scope and encryption details of XML documents. Compared with previous research works in element-wise encryption, the DSL is the only language which retains the syntax of XSL. For efficient and automatic consideration, an environment contains algorithm container, algorithm provider, algorithm lookup service, key provider and key manager are proposed. They are developed to support the securing tool which perform encryption and decryption of XML documents. In addition, we develop a DSL editor with a friendly graphic user interface to help users to generate DSL documents.

Section 1. Introduction

XML (extensible Markup Language) [1] is a markup meta-language recently standardized by the World Wide Web Consortium (W3C). While HTML was defined using only a small and basic part of the ISO-8879 SGML (Standard Generalized Markup Language) [2], XML is a sophisticated subset of SGML. It is designed to describe data using arbitrary tags and has recently emerged as the most relevant standardization effort in the area of document representation through markup languages in Web-based information systems. Main features of XML are related to the use of tags, for defining nested document structures. There are several classes of applications for XML: publishing, specification and data exchange.

As the intrinsic standardized property of XML document, it allows a convenient way to carry out data exchange between heterogeneous platforms among organizations via the Internet. Internet is a public network, and traditionally there has been little protection against unauthorized access to sensitive information and attacks. The objective of information security is to protect valuable information. There is a strong need for establishing a standard for information security within XML. A trivial way to secure a XML document is to employ the existing cryptography to encrypt an XML document as a whole. The receiver of an encrypted XML document then decrypts it with the appropriate key and algorithm [3,4,5]. However, some researchers thought that the usages of encrypting a whole XML document without dividing it up is quite limited. Subsequently, they proposed the element-wise encryption for XML document [6,7,8,9,10].

Within an XML source document, the data content is described by its corresponding element. In the element-wise encryption, the scope of encryption could be a whole element, some of the attributes of an element, or content of an element. Also, an attribute has two encryption selections: one is to only encrypt the content of it and the other is to encrypt both its name and content. An encrypted document includes raw and encrypted elements with encryption information of itself. The encryption information should contain at least location of the keys used for encryption, the type and location of algorithms used for encryption, the scope of encryption and so on. The receiver of the encrypted XML document then uses the attached encryption information to decrypt it accordingly. By defining a security tag and placing it around one or more elements in the XML source document, the securing tool can encrypt the
data content, including the embracing tags. The securing tool parses the secured XML document and recognizes the security tag to take appropriate actions to decrypt the contents of the security tag, if the appropriate key and algorithm is available. If proper authorization for the secure content is prohibited, the securing tool can only skip to the plain part of the XML source document. Figure 1.1 illustrates the element-wise encryption. Compared with the original document, only the two elements, “price” and “cardno”, are encrypted. Note that Figure 1.1 does not show the syntax of the encrypted XML document but demonstrates it abstractly.

![Figure 1.1: Element-wise encryption](image)

In the follows, we give some motivating examples which show the situations in which certain parts of an XML document need to be encrypted and the rest should be in clear text.

- **In publishing privileged document:** For example, a company could publish all employees’ personal data with element-wise encrypted XML documents. Although all the those documents are available to all users, the personal data are divided into several different portions with different encryptions. Then, a specific part can only be read by a limited number of persons who are capable of decrypting this part.

- **In routing document:** For example, in an e-commerce system, a customer sends an order form with element-wise encrypted XML document which contains the customer’s credit card information and related purchase details to the vendor. However, the credit card information is encrypted by the public key of credit card bank. The vendor can not read the credit card information but only the related information about the purchased goods. After he ships the goods to the customer, he has to route the document to customer’s credit card bank to obtain a clearance. In this case, although customer’s credit card information is embedded in the order form, only his credit card bank can read it.

In this paper, we address the issues of designing and implementing an efficient and automatic environment for securing XML document with element-wise consideration. The first part of this work is to study how to define the security information for an XML document. Following the XML concept of separating content from its meaning and its representation. It is clear that the encryption rule should be addressed in the same way: separate it from the content. We define the Document Security Language (DSL for short). The DSL works in separating the actual security description from the XML code. Especially, its syntax is completely the same with XSL[11]. There are some predefined XSL functions and XSL elements. DSL is defined by supplying some security related functions and elements to the original XSL. This has two advantages: (1) Users who are familiar with XSL can learn how to use DSL quickly and effortlessly; (2) The existing XSL software packages can be easily modified to a securing tool with element-wise encryption and decryption functionally for DSL. Based on our survey of previous works in element-wise encryption of XML, the DSL is the only security description language which still retains the complete syntax of XSL.
Secondly, based on the DSL, we construct an automatic and efficient security environment for XML. Generally speaking, the security information for encryption and decryption includes appropriate algorithms and keys. A **securing tool** executes encryption (or decryption) algorithms with the corresponding keys to encrypt (or decrypt) XML documents. As the keys are usually provided by authorized organizations[15,16], there must be a proper interface for a securing tool to download the keys. Also, for the consideration of efficiency, the key should not be downloaded repeatedly. In our infrastructure, the interface between a key provider and key manager is defined. A key provider is maintained by the authorized organization which generated the keys. A key manager is responsible to download keys from key providers as well as protect, store, and refresh the downloaded keys. Consider algorithms for encryption and decryption. If the users have to install the required algorithms in the securing tool manually, then the securing tool does not provide an automatic security environment. For this consideration, the algorithm should be able to be downloaded from the Internet and be executed in any platforms. Based on the above considerations, we design a protocol for automatic download of encryption algorithms. The operation of the protocol consists of several components including algorithm container, algorithm provider and algorithm lookup service. The securing tool communicates with the algorithm container which takes charge of discovering, loading, storing, and refreshing algorithms for encryption and decryption.

In the third, we design a DSL editor which is a tool to help users to generate DSL documents. Instead of writing DSL source code directly, the users could make use of the DSL editor to construct a DSL document. It is always a complicated task to write the DSL source code as the target XML document may contain many elements which are with different security requirements and each attribute of an element or the content of an element itself may have distinct security patterns. The users should specify the security details for each element which is going to be encrypted. It includes the scope of encryption in an element. A security pattern should include the detailed information of the corresponding key and the encryption algorithm. Our DSL editor is with a friendly graphic user interface for the user to specify the security information and then generates a compiled DSL document.

Remainder of the paper is organized as follows. Section 2 presents the construct and syntax of Document Security Language (DSL). Based on the definition of DSL, section 3 introduces how to support an automatic, efficient and integrated security environment with key manager, key provider, algorithm container, algorithm provider, and algorithm lookup service. Section 4 describes the construct of DSL editor. Section 5 concludes the paper and outlines future work.

**Section 2. Document Security Language**

The first part of this work is to study how to define the security information for an XML document. Following the XML concept of separating content from its meaning and its representation, it is clear that the encryption rule should be addressed in the same way: separate it from the content. We propose the Document Security Language (DSL for short). The DSL works in separating the actual security description from the XML code. See **Figure 2.1**. The **securing tool** is responsible for encrypting and decrypting XML documents. While the **securing tool** is encrypting an XML document, it reads the corresponding DSL document which contains all the details of the security information for the XML document including the scope of encryption, ways to get the keys, type and location of encryption algorithms and so on. Then, the securing tool interprets the DSL document and accordingly translates the original XML document to an encrypted XML document. In the reverse direction, i.e. decryption, the securing tool works in a similar way.
In the follows, we present the DSL. According to the functionality of DSL, it is used to specify the transformation between plain and encrypted XML document. In the XML community, there is also a transformation-related language called XSL [11], the W3C recommendation. The usage of XSL includes: (1) converting XML documents to, HTML and other formats; (2) publishing a large set of documents; (3) reorganize XML documents to create table of contents or other information; (4) extract information from XML documents, etc. We demonstrate that the syntax of XSL can be applied to the DSL as there is a similarity in their functionality, i.e. performing transformation for XML documents. There are some predefined XSL functions and XSL elements. DSL is an extension of the XSL by supplying some security related functions and elements to the XSL and retain the syntax of XSL. Compared with the works of [12,13], they defined new syntax and grammar for element-wise encryption. Although the SSML in [13] also employed XSL as a base for specifying security information, it still modified the syntax of XSL. It only retains the syntax of XSL template to match an element. However, there is no transformation element in SSML.

The advantages of retaining the syntax of XSL are twofold: (1) Users who are familiar with XSL can learn how to use DSL quickly and effortlessly; (2) The existing XSL software packages can be easily modified to a securing tool with element-wise encryption and decryption functionally for DSL.

Figure 2.2 shows an XML document. We use it as a running example to demonstrate the DSL.

```xml
<?xml version="1.0"?>
<transactions>
  <transaction>
    <name position="boss" sex="M">tony yao</name>
    <product type1="p01" type2="p02" type3="p03">computer</product>
    <price>499.00</price>
    <cardno>1234-5678-8765-4321</cardno>
  </transaction>
</transactions>
```

Figure 2.2: An XML document as a running example

A complete DSL document contains two parts. One is the security pattern section which collects all the security patterns applied to the target XML document. The other is referred to transformation description section which describes the transformation details.

Figure 2.3 shows an example of a security pattern section in a DSL document. It defines two security patterns named “pattern1” and “pattern2”. Each security pattern specifies the location, id, protocol of the key as well as the location, type, version, protocol of the algorithm. Note that the information defined in a security pattern is not limited to those that appear in Figure 2.3. We only list some important items.
Figure 2.4 is an example of a transformation description section in a DSL document. See line 5 to 9. It specifies to encrypt the whole element named “transaction/carno” with security pattern “pattern1”. The “value_of_encrypted_subtree” is an extension of XSL functions. It extracts the subtree in the matched node and then encrypts it with the specified security pattern. Combining the security pattern section in Figure 2.4 and the transformation description section in Figure 2.4 to a DSL, a securing tool transform the original XML document in Figure 2.2 to an encrypted XML document shown in Figure 2.5

```xml
<?xml version="1.0" ?>
<dsl:stylesheet xmlns:dsl="uri:dsl">
  <dsl:security_pattern id="pattern1">
    <key>
      <location>im.ncnu.edu.tw/segw/key</location>
      <id>ghhwang</id>
      <protocol>http</protocol>
    </key>
    <algorithm>
      <location>algserver.im.ncnu.edu.tw</location>
      <type>rsa-v2</type>
      <version>1.01a</version>
      <protocol>jini</protocol>
    </algorithm>
  </dsl:security_pattern>
  <dsl:security_pattern id="pattern2">
    <key>
      <location>im.ncnu.edu.tw/segw/key</location>
      <id>dakin</id>
      <protocol>http</protocol>
    </key>
    <algorithm>
      <location>algserver.im.ncnu.edu.tw</location>
      <type>rsa-v2</type>
      <version>1.01a</version>
      <protocol>jini</protocol>
    </algorithm>
  </dsl:security_pattern>
</dsl:stylesheet>
```

Figure 2.3: An example of a security pattern section in a DSL document

```xml
1   <dsl:template match="\">  
2     < ? x m l  v e r s i o n = " 1 . 0 "  ? >  
3      <dsl:apply-templates/>  
4   </dsl:template>  
5   <dsl:template match="transaction/carno">  
6     <encrypted-element scope="content_of_element" pattern="pattern1">  
7       <dsl:value_of_encrypted_subtree/>  
8     </encrypted-element>  
9   </dsl:template>
```

Figure 2.4: An example of a transformation description section in a DSL document
In addition to encrypt an element in a whole, the DSL can specify to encrypt only the content or some of the attributes of an element. Also, an attribute has two encryption selections: one is to only encrypt the content of it and the other is to encrypt both its name and content. In the follows, we use an example to illustrate the flexibility of the DSL. See Figure 2.6. Line 8 and 9 specify that the encryption scope contains the attribute named “position” and the content of element “transaction/name”. According to the definition of DSL, only the content of the “position” attribute will be encrypted. The corresponding security patterns are “pattern1” and “pattern2” respectively. The “value_of_encrypted_node” is an extension of XSL elements, which encrypts the current node according to the security information designates in the attributes of “encrypted-element” (Line 8 and 9 in this case). Moreover, line 15 and 16 specify that the encryption scope contains the first and third attributes with security pattern “pattern2” and “pattern1” respectively. Line 15 only shows the sequential number of attributes to be encrypted. According to the definition of DSL, both the name and the content of the first and third attributes of node “transaction/product” will be encrypted.
See the Figure 2.7. It is the encrypted XML document of the original XML document in Figure 2.2 according to the DSL of Figure 2.3 and Figure 2.6. Note that only the content of the attribute “position” in element “transaction/name” is encrypted. However, both the name and the content of the first and third attributes of element “transaction/product” are encrypted.

![Figure 2.7: An example of an encrypted XML document](image)

We present the syntax and capability of the DSL in this section. Subsequently, we present the way to support automatic and efficient encryption with DSL.

Section 3. Efficient and Automatic Encryption with DSL

The key point to an efficient and automatic DSL-based securing tool is with two major considerations: one is about the way for a securing tool to obtain the keys and the other is how to get the appropriate encryption and decryption algorithm automatically installed and invoke it correctly. In the security pattern section of a DSL document, each security pattern specifies the location, id, and protocol of the key as well as the location, type, version, protocol of the algorithm. In the follows, we present how to cope with these two problems.

Section 3.1 Algorithm Container, Algorithm Provider, and Algorithm Lookup Service

As presented in Figure 2.1, the securing tool reads the DSL document to obtain the security information. A security pattern in a DSL document describes the key and algorithm for a designated encryption. In addition to install the required encryption algorithms manually, we think there must be a mechanism for the securing tool to obtain the required algorithms automatically, especially, without any user’s intervention. Also, the downloaded algorithms should be able to be executed in heterogeneous platforms. Refer to Figure 3.1. There are three components in our infrastructure: algorithm provider, algorithm lookup service, and algorithm container. An organization which...
implements encryption algorithms makes use of an algorithm provider to supply algorithms to the algorithm lookup service. The encryption and decryption algorithms are implemented in Java object and dispatched to the algorithm lookup service. In addition to the binary format of the Java object, the attributes which annotate the object attached to this object. The attributes of an algorithm object should include the name, type, version, expiration date, provider and so on. An algorithm lookup service is a long-run server which keeps track all the algorithm objects and their attributes that many algorithm providers have registered. An algorithm container then downloads the required algorithms defined in DSL automatically for the securing tool. In addition to download the required algorithms for the securing tool, an algorithm container also stores the downloaded algorithm objects as well as their attributes for future use and refreshes the stored algorithm objects periodically. As the securing tool needs to invoke an encryption algorithm specified in a DSL document, it first asks the algorithm container for it. If the target encryption algorithm is not available in the algorithm container, the algorithm container then connects to the algorithm lookup service according to the location of the encryption algorithm specified in the DSL document. It looks up the algorithm object in the algorithm lookup service and downloads the required algorithm. The downloads algorithm is then stored in the algorithm container for future use. Moreover, the algorithm container may renew the algorithm object periodically according to the attributes attached in the algorithm object.

We use the Jini architecture [14] as the protocol for the communication between algorithm container, algorithm lookup service and algorithm provider. It includes the discovery, registration and lookup protocol for encryption algorithm objects.

![Automatic algorithm download mechanism](image)

**Figure 3.1:** Automatic algorithm download mechanism

### Section 3.2 Key Manager and Key Provider

According to the syntax of the DSL, different fragments of an XML document may be encrypted with different keys and algorithms. For constructing an efficient and automatic architecture of a securing tool, there are two components in our infrastructure for managing the download and lookup of keys: the key manager and the key provider. Take a look at Figure 3.2. A key provider is a long-run server maintained by an key authoring organization [15,16] that generates keys for users. The public keys are published in the key provider. In addition to the binary format of a key, the related information of a key includes the owner’s information, expiration date, corresponding algorithm details is also in a dictionary in the key provider. When a securing tool needs a key to encrypt or decrypt the data in an XML document, it first asks the key manager for it. The key manager will check if the key is stored in its key pool. As the key is not available, the key manager connects to the appropriate key provider. Then, it looks up the dictionary of the key provider and download the appropriate key accordingly via the HTTP protocol. The securing tool could get a key without resorting to a key provider in a remote site if the supporting key manager is with a copy in its key pool. This provides an efficient and automatic way to download and manager the keys.

The key manager is to support the securing tool to download the keys, store the keys and refresh the
keys when a key is expired.

Figure 3.2: Automatic key download mechanism

Section 3.3 An Integrated Environment for XML Security

Figure 3.3 shows an integrated environment for efficient and automatic encryption and decryption for XML documents. It includes the securing tool, algorithm container, key manager, algorithm provider, algorithm lookup service, and key provider. While the encryption (or decryption) is performed, the securing tool reads a plain (or encrypted) XML and its corresponding DSL documents. By analyzing the security pattern section in the DSL document, it sends messages to the algorithm container and key manager for requesting required keys and algorithms. For the efficient consideration, the algorithm container and key manager will first check to see if there are already copies of the required keys and algorithms in their local store. If there are some algorithms that are not available in the local site, the algorithm container will connect to the appropriate algorithm lookup service to download the required algorithms automatically. Similarly, the key manager connects to a key provider only when the desired key is not available in its key pool.

Figure 3.3: The integral environment for XML security

Section 4 DSL Editor

An XSL document is also an XML document. As we have mentioned in Section 2, the DSL
complies with the syntax of XML. That is, an experienced XSL programmer should be able to code a DSL document easily. However, coding a DSL document is quite tedious as each element and attribute may have different security patterns, i.e. with different key and algorithm specifications. Based on the infrastructure mentioned in Section 3, we design an DSL editor to help the user with the writing of DSL documents. Figure 4.1 illustrates the work of our DSL editor.

![Diagram of DSL editor](image)

**Figure 4.1: The operation of a DSL editor**

The goals of XML editor, XSL editor, and DSL editor are the same: to provide a convenient tool to user. However, the operation and the structure of a DSL editor is more complicated then XML and XSL editor. Basically, a DSL editor first reads an XML or a DTD document to obtain the tree structure of the target XML document. It also communicates with the algorithm container and key manager to get the information of available algorithms and keys. In its graphic interface, the user sets up the security pattern for the elements and attributes. The user can generate a security pattern by selecting the keys and algorithms from the information of available algorithms and keys from the algorithm container and key manager or input the related information of keys and algorithms manually. Table 4.1 summaries the functions of XML, XSL and DSL editors.

<table>
<thead>
<tr>
<th>Editor tool</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>XML Editor</td>
<td>A graphical user interface which allows users to edit XML document, configure the parameters, and view XML.</td>
</tr>
<tr>
<td>XSL Editor</td>
<td>A XSL stylesheet editor incorporates trace function and provides assistance with writing select and match expressions. It integrates XSL Trace with the Visual Transform tool.</td>
</tr>
<tr>
<td>DSL Editor</td>
<td>According to the target XML, and information in the algorithm container and key manager to form a graphic interface to allow the user to generate DSL document without coding the details syntax of DSL.</td>
</tr>
</tbody>
</table>

**Table 4.1: Description of XML, XSL and DSL editors**

In the follows, we use the graphic user interface of our DSL editor to illustrate its function. In Figure 4.2, the users inputs the file name of the XML document (or its DTD document). The DSL editor then shows the tree structure of the target document in the window frame.
In the figure 4.3, the user clicks and selects an element (titled “name” in this case) and constructs its security pattern. He is now selecting the key from the key list of the security pattern of element “name”. Note that the key list is obtained from the key manager. As there is no desired key in the key list obtained from the key manager, the users can input a key manually by keying in the id, protocol and location of the key.
Figure 4.3: **DSL editor:** Choose an element and construct its security pattern

Following the Figure 4.3. The user chooses the desired algorithm in Figure 4.4. It is similar to the process of selecting and inputting a desire key.

![Figure 4.3: DSL editor](image)

Figure 4.4: **DSL editor:** Choose an element and construct its security pattern

Figure 4.5 illustrates the screen that the user are selecting the scope of encryption.

![Figure 4.5: DSL editor](image)
Section 6 Conclusion and Future Work

In this paper, we address the issues of designing and implementing an efficient and automatic environment for securing XML document with element-wise consideration. First of all, we define the syntax the Document Security Language which is able to describe the scope and security pattern of the encryption. DSL works by supplying some security related functions to the original XSL functions. It has two advantages: (1) Users who are familiar with XSL can learn how to use DSL quickly and effortlessly; (2) The existing XSL software packages can be easily modified to a securing tool with element-wise encryption and decryption functionally for DSL.

Secondly, based on the DSL, we construct an automatic and efficient security environment for XML. Generally speaking, the security information for encryption and decryption includes at least the appropriate algorithms and keys. The securing tool communicates with algorithm container and key manager to obtain an efficient and automatic style for the download, management and invocation of keys and encryption algorithms. The discovery and lookup protocol of algorithm lookup service and algorithm provider supports the operation of algorithm container. Similarly, key manager needs the key providers to work out. In the third, we design a DSL editor which is a tool to help users to construct a DSL document. In stead of writing DSL source code directly, the users could make use of the DSL editor to construct a DSL document.

In addition to work on implement the whole infrastructure, there are still some research topics. For example, we are study to apply the element-wise encryption environment developed in this paper to XML database, XML-based E-commerce and workflow management system, etc.

Reference

13. Paul Brandt and Frederik Bonte, “Towards secure XML”.