

## Short Paper

---

# A Secure Query Language for XML Documents<sup>\*</sup>

TAO-KU CHANG AND GWAN-HWAN HWANG<sup>†</sup>

*Department of Computer and Information Science  
National Dong Hwa University  
Hualien, 970 Taiwan*

<sup>†</sup>*Department of Computer Science and Information Engineering  
National Taiwan Normal University  
Taipei, 106 Taiwan  
E-mail: ghhwang@csie.ntnu.edu.tw*

The intrinsic standardized property of an XML document provides a convenient way to carry out data exchanges between heterogeneous platforms among organizations via the Internet. The Internet is a public network, and traditionally there has been little protection against unauthorized access to sensitive information, and attacks. Although the W3C proposed the XQuery language [3], which is designed to be broadly applicable across all types of XML data sources, this language does not provide a security mechanism in its query expressions. In this paper, we propose a new XML query language, called the **secure XML Query (sXQuery)** language. sXQuery is derived from XQuery, and reinforced with a security mechanism; sXQuery combines the specification ability of both the XQuery language and the document security language which is designed to specify the scope and encryption details of XML [9, 11]. The user can specify the query and corresponding encryption details at the same time, that is, in the same sXQuery document. We have designed an sXQuery editor which enables users to generate sXQuery documents without having to write sXQuery source codes directly. Also, we present a scheme to implement an sXQuery engine by using the existing XQuery engine.

**Keywords:** XML, security, database, XQuery, sXQuery, DSL

## 1. INTRODUCTION

XML (extensible markup language) [1] is a markup meta-language that was standardized by the World Wide Web Consortium (W3C, <http://www.w3.org>). While HTML (hypertext markup language) was defined using only a small and basic part of the standard generalized markup language (ISO-8879 SGML) [2], XML is a sophisticated subset of SGML. It was designed to describe data using arbitrary tags and has emerged as the most relevant standardization effort in the area of document representation through markup languages in Web-based information systems.

---

Received January 12, 2007; revised June 26, 2007; accepted August 8, 2007.

Communicated by Chin-Laung Lei.

<sup>\*</sup> The preliminary result of this research was presented in International Conference on Internet Computing, 2002, Las Vegas, Nevada, USA. This work was supported in part by the National Science Council of Taiwan, R.O.C. under grants No. 94-2213-E-003-006 and 95-2221-E-003-007.

<sup>†</sup> Corresponding author.

The W3C proposed the XQuery language [3], which is designed to be broadly applicable across all types of XML data sources. The mission of XQuery is to provide flexible query facilities to extract data from real and virtual documents on the Web, therefore finally providing the needed interaction between Web and database worlds. However, XQuery does not provide any security mechanism in its query expressions. As the intrinsic standardized property of an XML document, it allows a convenient way to carry out data exchanges between heterogeneous platforms among organizations via the Internet. The 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, and there is a strong need for establishing a standard for information security within XML.

A trivial way to secure an XML document is to employ existing cryptography techniques to encrypt an XML document as a whole. The receiver of an encrypted XML document then decrypts it with the appropriate key and algorithm [4-6]. However, some researchers considered that encrypting an entire XML document without dividing it up limits its usage. They therefore proposed the element-wise encryption of XML documents [7-15]. Based on some work on XML element-wise encryption, the W3C delivered a recommendation specification for XML encryption [15]. The document specifies a process for encrypting data and representing the result in XML. The data may be arbitrary data, an XML element, or the content of an XML element. The encryption and signature standards proposed by W3C set up the format for the secured XML documents. However, they do not include the ability for the programmer to specify how to encrypt and sign his or her XML documents. To overcome this problem, we proposed an operational model which is with a way for the programmer to specify the security detail of XML documents. Do achieve this, we defined a language called document security language (DSL) [9, 11]. A DSL document defines the transformation description for encrypting, decrypting, embedding and verifying signatures. It offers a security mechanism that integrates *element-wise encryption* and *temporal-based element-wise digital signatures*. Also, because the syntax of the element EncryptedData in the XML encryption standard makes it impossible to extend it to handle attribute encryption, the DSL supports a type of element-wise encryption that is more general: the scope of encryption (or encryption granularity) could be a whole element, some of the attributes of an element, or the content of an element; an attribute has two possible types of encryption: one is to only encrypt the value of it and the other is to encrypt both its name and value. The secured document produced by the DSL securing tool can be compatible with the XML encryption and digital signature standard in case that the attribute encryption is not applied.

Fig. 1 illustrates the relationship between XML, DSL, and the *DSL securing tool*. Fig. 1 (a) shows the process of encrypting and embedding digital signatures. The encryption and digital signature details are stored in a DSL document comprising  $D_P$ ,  $D_T$ , and  $D_{Sig}$ .  $D_P$  is the security pattern definition that specifies the combination of security algorithms and encryption and decryption keys, the transformation description definition  $D_T$  specifies the actual data transformation of the element-wise encryption, and  $D_{Sig}$  specifies how to embed digital signatures in the resulting XML document. The target XML document that is ready to be encrypted and signed (or signed) is  $X$ . The DSL securing tool reads, parses, and analyzes  $D_P$ ,  $D_T$ ,  $D_{Sig}$ , and  $X$ , and then generates  $X_s$ , and  $D_P$ .  $X_s$  is still an XML document, but some of its elements contain ciphertexts which are translated by the

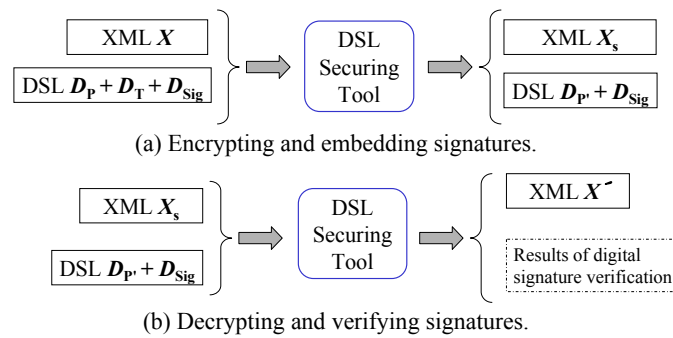


Fig. 1. XML, DSL, DSL securing tool, and transformation.

DSL securing tool according to the encryption details recorded in  $D_P$  and  $D_T$ . In addition to the encrypted elements,  $X_s$  also contains signatures that are embedded by the DSL securing tool. Each signature signs a portion of the data in  $X$ . It should be noted that both  $D_P$  and  $D_{P'}$  may actually contain different information:  $D_P$  holds information describing how to encrypt  $X$ , whereas  $D_{P'}$  should include details of how to decrypt  $X_s$ . In addition, we have developed a DSL editor with a friendly graphic user interface to make it easier for users to generate DSL documents [11].

XQuery does not include a security mechanism in its query expressions, but we can employ DSL to achieve this (see Fig. 2 (a)). It takes two steps to obtain secured (or *encrypted*) XML documents from a database:

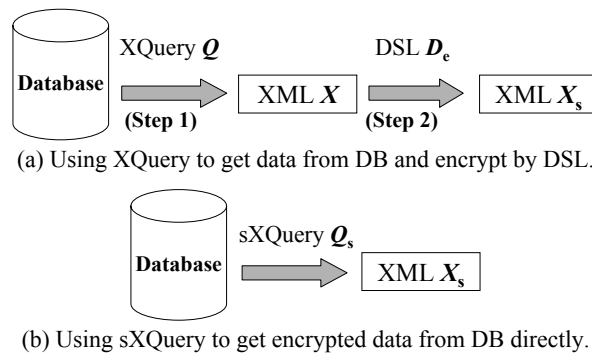


Fig. 2. XQuery and sXQuery.

**Step 1:** The query details are specified by an XQuery document,  $Q$ . Assume that we obtain an XML document  $X$  from the database transaction  $Q$ .

**Step 2:** A DSL securing tool transforms  $X$  to  $X_s$  according to a DSL document,  $D_e$ , which describes the element-wise encryption details corresponding to  $X$ .

It can be easily seen that this two-step process has some drawbacks. First,  $X$  will be discarded after step 2 since our goal is to obtain  $X_s$ , and this is associated with at least

two overheads: (1) writing  $X$  out and reading it back in, and (2) an additional pass over  $X$ . Second,  $X$  is no longer secured or under the protection of the database server, which decreases the security of the query transaction. If step 2 is executed in a machine other than the database server,  $X$  must be transferred to that machine to perform step 2 and it is obvious that  $X$  is unprotected during this transfer. Third, the XQuery document  $Q$  and DSL document  $D_e$  are two separate files, which causes difficulties in writing them. There is good evidence to show that the programmer should attempt to coordinate codes in  $Q$  and  $D_e$ . Moreover, elements that do not need to appear in  $X_s$  may need to be stored in  $X$  in order to obtain a specific security requirement. For instance, some of the elements in  $X$  are only used to locate the elements which should be encrypted and stored in  $X_s$ . We use an example to illustrate this situation in chapter 5 of [16].

In this paper, we propose a new XML query language, denoted as the **secure XML Query (sXQuery)** language. sXQuery is derived from XQuery, and reinforced with a security mechanism. First, sXQuery adds some *security information manipulation expressions* which do not exist in XQuery. Second, sXQuery extends the data manipulation expression of XQuery by allowing the addition of an *sXQuery transformation descriptor* to the data manipulation expression of XQuery. Actually, sXQuery combines the specification ability of both XQuery and DSL. The user can specify the query and corresponding encryption details at the same time; *i.e.*, in the same sXQuery document (see Fig. 2 (b)). The user specifies the query transaction in an sXQuery document,  $Q_s$ .  $X_s$  is obtained after only one step. It is obvious that  $Q_s$  contains information which covers  $Q$  and  $D_e$ . This means that the three drawbacks existing in the two-step process in Fig. 2 (a) disappear.

The remainder of the paper is organized as follows. Section 2 presents the syntax of sXQuery. Section 3 describes the architecture of an sXQuery database and sXQuery editor. Sections 4 and 5 present our implementation and experimental results. Section 6 concludes the paper.

## 2. THE SYNTAX OF sXQuery

Because of the drawbacks of the two-step process (Fig. 2 (a)) mentioned in section 1, we propose an XML query language (denoted sXQuery) in this paper. As also mentioned in section 1, sXQuery is derived from XQuery and reinforced with security mechanism; sXQuery combines the specification ability of both the XQuery language and the DSL. sXQuery is a superset of XQuery with two reinforcements to the syntax:

1. sXQuery adds some *security information manipulation expressions* which do not exist in XQuery. It allows the user to define, delete, and query the required keys, algorithms, security patterns, and transformation templates. The defined keys, algorithms, security patterns, and transformation templates for encryption and decryption can be stored in the database. They can also be deleted from the database.
2. sXQuery extends the data manipulation expression of XQuery by allowing the addition of the *sXQuery transformation descriptor* to the data manipulation expression of XQuery. The enhanced data manipulation expression of sXQuery can instruct the database to process a data query and encryption in a single transaction.

We present the security *information manipulation expression* in section 2.1. The syntax of *sXQuery transformation descriptor* and its usage are given in section 2.2.

## 2.1 Security Information Manipulation Expression of sXQuery

The basis of the security information manipulation expression of sXQuery is the ability to define keys, algorithms, security patterns, and transformation templates corresponding to the encryption and decryption method. sXQuery adopts the syntax of the key, algorithm, and security pattern definitions of DSL [9, 11]. The security information manipulation expression of sXQuery can be classified into four categories. The first type is the *add* operation, which adds some key, algorithm, security pattern, and transformation template definitions to the database for future use. We show it in BNF form<sup>1</sup> in Syntax 1. The **Key Definition**, **Algorithm Definition**, and **Security Pattern Definition** are BNF nonterminal symbols, and defined in [11]. The **Key Definition** represents the syntax of key definition which will be referenced in the security pattern definition. **Algorithm Definition** gives the syntax of the algorithm definition which will also be referenced in security pattern definition. According to the requirements of element-wise encryption, each encrypted section may have different security patterns. The **Security Pattern Definition** contains the encryption or decryption key, algorithm, and other related information. The **Transformation Template Definition**<sup>2</sup> specifies the way to match XML elements, and the related security patterns to transform the matched XML elements.

**Syntax 1:** The add operations of the *sXQuery* security information manipulation expression.

```

Add Key Statement →
  $$XQuery_ADD Key Definition
Add Algorithm Statement →
  $$XQuery_ADD Algorithm Definition
Add Security Pattern Statement →
  $$XQuery_ADD Security Pattern Definition
Add Transformation Template Statement →
  $$XQuery_ADD
  Transformation Template Definition
Transformation Template Definitions →
<sXQuery:template name="Transformation Template Name" match="xpath">
  <sXQuery:value-of-encrypted-node
    scope="Scope Description{,Scope Description}"
    pattern="Security Pattern Name{,Security Pattern Name}"/>
</sXQuery:template>

```

The syntax of the transformation template definition of sXQuery is different from the transformation template of DSL. Because DSL is designed to be compatible to the syntax of XSLT [17], it must include some XSLT elements such as “apply-templates.” Since sXQuery is designed to conform to XQuery, it does not have to involve XSLT elements. For example, see the sXQuery statements shown in Example 1: the key “ghhwang.pub” and “ghhwang.prv” are first defined, then algorithms “http://www.w3.org/2001/04/xmlenc#tripleledes-cbc” and “http://www.w3.org/2001/04/xmlenc#rsa-1\_5” are defined,

<sup>1</sup> See [23] for more information about the BNF representation.

<sup>2</sup> The definition of this BNF nonterminal is shown in [11].

and finally a security pattern “P1” is defined. It must be noted that the “P1” use “encrypted\_format=“DSL”” to apply the syntax of encrypted XML document defined in [11] and use “encrypted\_format=“W3C”” to apply XML encryption syntax and processing proposed by W3C [15].

**Example 1:** Some *ADD* operations of *sXQuery*.

```

$sXQuery_ADD
<key_definition key_link="enc-ghhwang">
  <key_name>ghhwang.pub</key_name>
</key_definition>
$sXQuery_ADD
<key_definition key_link="dec-ghhwang">
  <key_name>ghhwang.prv</key_name>
</key_definition>
$sXQuery_ADD
<algorithm_definition algorithm_link="agltripleDES-cbc" use="SECURITY">
  <algorithm_id>
    http://www.w3.org/2001/04/xmlenc#tripleDES-cbc
  </algorithm_id>
</algorithm_definition>
$sXQuery_ADD
<algorithm_definition algorithm_link="aglrSA-v15" use="SECURITY">
  <algorithm_id>
    http://www.w3.org/2001/04/xmlenc#rsa-1_5
  </algorithm_id>
</algorithm_definition>
$sXQuery_ADD
<security_pattern name="P1" encrypted_format="W3C">
  <key_information>
    <encryption_key>
      <key_definition key_link="enc-ghhwang"/>
    </encryption_key>
    <decryption_key>
      <key_definition key_link="dec-ghhwang"/>
    </decryption_key>
  </key_information>
  <security_algorithm using_key_encryption="YES">
    <algorithm_definition algorithm_link="agltripleDES-cbc"/>
    <key_encryption_algorithm>
      <algorithm_definition algorithm_link="aglrSA-v15"/>
    </key_encryption_algorithm>
  </security_algorithm>
</security_pattern>

```

We show an example of the transformation template definition in Example 2. The first transformation template applies to the elements of a queried XML document with pathname “\*/name”. In each matched element, the value of attribute “rank” is encrypted with security pattern “P1,” and the content of the element is also encrypted with security pattern “P1.” In the second transformation template, each element with pathname “\*/product” will be matched. The first and the third attributes will be all encrypted with security patterns “P1”. Note that the value and the name of the attributes are encrypted as a whole. In the third transformation template, the subtrees rooted in elements with pathname “\*/cardinfo” are extracted and encrypted with security pattern “P1.”

**Example 2:** Three transformation template definitions of *sXQuery*.

```

$sXQuery_ADD
<sXQuery:template name="TName" match="name">
  <sXQuery:value-of-encrypted-node
    scope="attribute_name=rank, content" pattern="P1,P1"/>
</sXQuery:template>
$sXQuery_ADD
<sXQuery:template name="TProduct" match="product">
  <sXQuery:value-of-encrypted-node
    scope="attribute_index=1,attribute_index=3" pattern="P1,P1"/>
</sXQuery:template>
$sXQuery_ADD
<sXQuery:template name="TCardinfo" match="cardinfo">
  <sXQuery:value-of-encrypted-node scope="element" pattern="P1"/>
</sXQuery:template>

```

The second type of *sXQuery* security information manipulation expression is the *delete* operation. It is used to *remove* the key, algorithm, or security pattern definition from the database.

**Syntax 2:** The delete operations of the security information manipulation expression of *sXQuery*.

```

Delete Key Statement →
  $sXQuery_Delete_Key
  Key Select Expression
  [Delete_Related_Security_Patterns]
  Key Select Expression → Where Key Select {Boolean Key Select Expression}
  Key Select → Key_Link="Key Link" | Key_Name="Name of Key"
  Boolean → And | Or
Delete Algorithm Statement →
  $sXQuery_Delete_Algorithm
  Algorithm Select Expression
  [Delete_Related_Security_Patterns]
  Algorithm Select Expression → Where Algorithm Select {Boolean Algorithm Select Expression}
  Algorithm Select →
    Algorithm_Link="Algorithm Link" | Algorithm_Name="Name of Algorithm"
Delete Security Pattern Statement →
  $sXQuery_Delete_Security_Pattern
  Security Pattern Select Expression
  [Delete_Related_Keys]
  [Delete_Related_Algorithms]
  Security Pattern Select Expression → Where Security Pattern Select
  {Boolean Security Pattern Select Expression}
  Security Pattern Select → Security_Pattern_Name="Security_Pattern_Name"
Delete Transformation Template Statement →
  $sXQuery_Delete_Transformation_Template
  Transformation Template Select Expression
  [Delete_Related_Keys]
  [Delete_Related_Algorithms]
  [Delete_Related_Security_Pattern]
  Transformation Template Select Expression →
    Where Transformation Template Select
    {Boolean Transformation Template Select Expression}
  Transformation Template Select →
    Transformation_Template_Name="Transformation_Template_Name"

```

The following *sXQuery* statement deletes the security pattern “P1” from the database. It also instructs the database to remove the used keys which are defined in “P1.”

**Example 3:** Delete statements.

```
$sXQuery_Delete_Security_Pattern
  Where Security_Pattern_Name="P1"
  Delete_Related_Keys
```

The third type of sXQuery security information manipulation expression is the *select* operation. It is designed to allow the user to look up the stored key, algorithm, and security pattern in the database. The syntax is shown in Syntax 3.

**Syntax 3:** The select operations of the security information manipulation expression.

```
Select Key Statement→
  $sXQuery_Select_Key
  Key Select Expression
Select Algorithm Statement→
  $sXQuery_Select_Algorithm
  Algorithm Select Expression
Select Security Pattern→
  $sXQuery_Select_Security_Pattern
  Security Pattern Select Expression
Select Transformation Template→
  $sXQuery_Select_Transformation_Template
  Transformation Template Select Expression
```

## 2.2 Transformation Descriptor of sXQuery

The principal forms of XQuery expressions include the path expression, element constructor, FLWOR expression, expressions involving operators and functions, conditional expression, quantified expression, and expressions that test or modify data types [3]. The sXQuery extends the syntax of XQuery by allowing the addition of sXQuery transformation descriptors to the XQuery expression to specify the way to match and transform (or encrypt) XML elements.

With the sXQuery transformation template definition shown in Syntax 1, we define the syntax of *sXQuery transformation descriptor*. Its BNF form is shown in Syntax 4.

**Syntax 4:** *sXQuery* transformation descriptor.

```
sXQuery Transformation Descriptor→
  $sXQuery_Secure
  {Transformation Template Name {Transformation Template Name}}
```

The first form of the extension of XQuery in sXQuery is to append the sXQuery transformation descriptor to the *path expression* of XQuery. The path expression of XQuery is based on the syntax of [18], which is a notation for navigating along “paths” in an XML document.

Consider the following XML document:

**XML Document 1** “*transactions.xml*”

```
<?xml version="1.0"?>
<transactions>
  <transaction>
    <name rank="boss" sex="M">tonyyao</name>
    <product type1="p01" type2="p02" type3="p03">computer</product>
```

```

<price>499.00</price>
<cardinfo>
  <cardtype>visa</cardtype>
  <cardlevel>g</cardlevel>
  <cardno>1234-5678-8765-4321</cardno>
  <cardowner>
    <name>tony yao</name>
    <id>m123456789</id>
  </cardowner>
</cardinfo>
</transaction>
</transactions>

```

The user can use the following XQuery statement to retrieve data from “transactions.xml” in the XML Document 1:

```
doc("transactions.xml")//*/transaction
```

The sXQuery statement that adds a transformation descriptor is shown below. The sXQuery transformation descriptors are underlined.

```

doc("transactions.xml")//*/transaction $sXQuery_Secure{TCardinfo}
doc("transactions.xml")//*/transaction $sXQuery_Secure{TName TProduct}

```

Note that **TCardinfo**, **TName**, and **TProduct** are transformation template definitions defined in Example 2. The query result of the above sXQuery statements are shown in chapter 5.1.2 [16].

The second form of extension is applied to the *element constructor* of XQuery. Similar to the extension of the path expression, this extension appends the sXQuery transformation descriptor to the element constructor of XQuery (see the following example). The transformation descriptor **\$sXQuery\_Secure{TName}** specifies the way to transform (or to encrypt) the element constructor of “*emp*”:

```

<emp empid="12345">
  <name>John Smith</name>
  <job>Anthropologist</job>
</emp> &sXQuery_Secure{TName}

```

The third form of extension is applied to the FLWOR expression of XQuery. A FLWOR expression binds values to one or more variables and then uses these variables to construct a result. A FLWOR expression may include the path expression or element constructor.

We show two examples. In the first example, the sXQuery transformation descriptor is appended to the path expression of an FLWOR expression:

```

FOR $b IN doc("bin.xml")//book
WHERE $b/publisher="Morgan Kaufmann"
AND $b/year="1998"
RETURN $b $sXQuery_Secure{TID BName}

```

For the second example, we show to append the sXQuery transformation descriptor to the element constructor of an FLWOR expression.

```

FOR $p IN distinct(doc("bib.xml")//publisher)
Let $a := avg(doc("bib.xml")//book[publisher=$p/price])
RETURN
  <publisher>
    <name>{$p/text()} </name>
    <avgprice>{$a}</avgprice>
  </publisher> $$XQuery_Secure{Tprice TID}

```

Other kinds of extension are applied to the conditional and quantified expression of XQuery. The extension is very similar to that of the FLWOR expression, since conditional and quantified expressions employ the path expression and element constructor to generate query results. The transformation descriptor is appended to the path expression or the element constructor. We show how to append the sXQuery transformation descriptor to the element constructor of a conditional expression in following sXQuery statement:

```

FOR $h IN //holding
RETURN
  <holding>
    {$h/title,
     IF ($h/@type="Journal")
     THEN $h/editor
     ELSE $h/author
    }
  </holding> $$XQuery_Secure{Teditor Tauthor}

```

The following example shows the extension of sXQuery to quantified expressions:

```

FOR $b IN //book
WHERE EVERY $p IN $b/para SATISFIES contains($p, "sailing")
RETURN $b/title $$XQuery_Secure{TID}

```

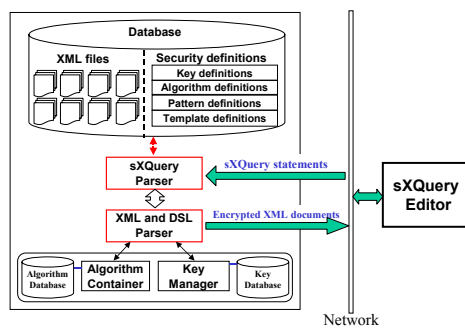
### 2.3 To Decrypt XML Documents Secured by sXQuery Database

XML documents that are secured (or encrypted) by sXQuery database are decrypted by following the transformation protocol of DSL [11]. According to the encryption rule of DSL, all the encrypted XML elements are transformed to become XML elements with a tag named *encrypted-element*. The corresponding security patterns and scopes are stored in the attributes of this tag. This means that the secured XML documents obtained from an sXQuery database can be decrypted by the DSL securing tool, or by any other tool that understands the transformation protocol of DSL. See [11] for details of the transformation protocol.

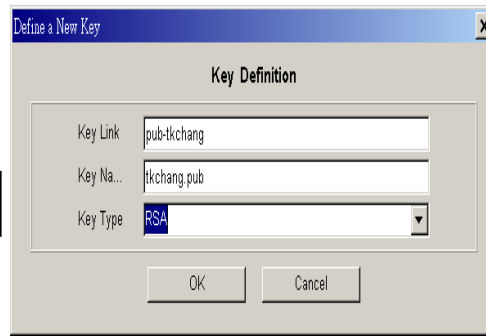
As we have mentioned in section 1, the secured document produced by the DSL securing tool can be compatible with the XML encryption and digital signature standard in case that the attribute encryption is not applied. The XML document secured by sXQuery database can conform to the form defined in [15].

## 3. THE ARCHITECTURE OF sXQuery DATABASE AND sXQuery EDITOR

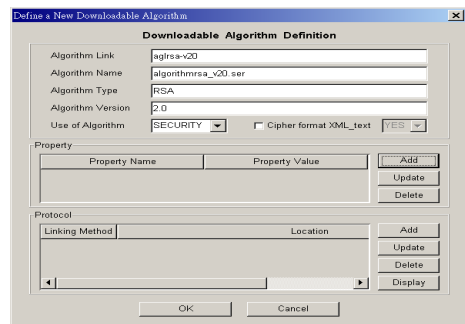
Fig. 3 (a) shows the architecture of the sXQuery database. The query transaction made by the user is first sent to the *sXQuery parser*. The sXQuery parser analyzes the



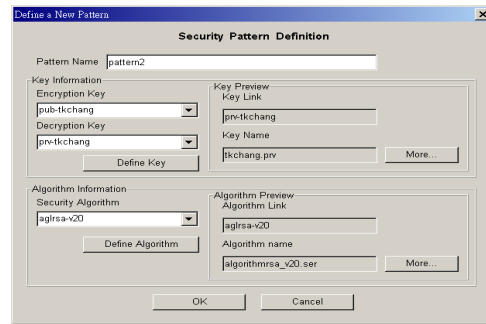
(a) The architecture of the sXQuery database.



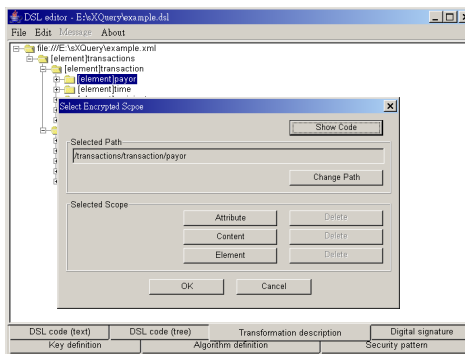
(b) sXQuery editor: making the key definition.



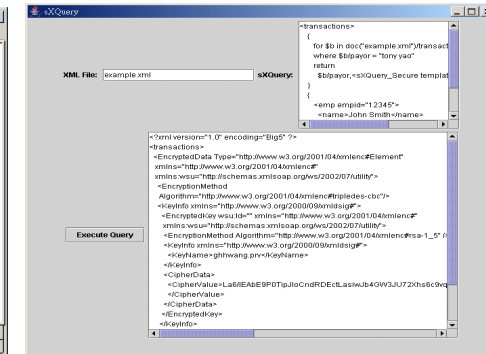
(c) sXQuery editor: making the algorithm definition.



(d) sXQuery editor: making the pattern definition.



(e) sXQuery editor: making the transformation template definition.



(f) sXQuery editor: writing and submitting sXQuery statements.

Fig. 3. The Architecture of sXQuery database and sXQuery editor.

sXQuery statements and retrieves XML elements from the XML files in the database. The queried XML elements and the associated security definitions are then delivered to the *DSL parser*. Subsequently, the DSL parser encrypts some of the received XML elements according the transformation descriptors in the sXQuery statements.

As the sXQuery statement is prepared, the user simply sends it to the sXQuery database via the network, and then an XML document with some secured XML elements is returned automatically. However, coding an sXQuery document is quite tedious since

each element and attribute may have different security patterns; *i.e.*, with different key and algorithm specifications. Also, different XML elements can have different transformation templates. We therefore designed an sXQuery editor to help the user to write sXQuery documents. Generally speaking, the sXQuery editor is very similar to the DSL editor we presented in [9, 11]. However, in addition to including the DSL editor's basic function of editing security definitions—including key, algorithm, security pattern, and transformation template definitions—the sXQuery editor is able to (1) obtain security definitions from the specified sXQuery database, (2) send security definitions to the database, and (3) delete security definitions from the database.

Figs. 3 (b)-(f) illustrate the work performed by our sXQuery editor. The end goal of XML, XSLT, DSL, and sXQuery editors is the same: to provide a convenient tool for the user. However, the operation and the structure of an sXQuery editor is more complicated than the other types of editor. Figs. 3 (b)-(e) show how to use the sXQuery editor to edit the key, algorithm, security pattern, and transformation template definitions, respectively. The user can also submit these definitions to database. After the user has defined the required transformation templates, he or she can easily write sXQuery query expressions (see Fig. 3 (f)).

#### 4. IMPLEMENTATION

Our goal is to construct a prototype to demonstrate the feasibility of the sXQuery rather than implement a high-performance sXQuery engine, which we achieve by reusing the existing XQuery engine, of which there are many implementations [19-21]. The operation of the implemented sXQuery engine involves the generation of some intermediate documents and some extra passes over these documents. However, the reuse of the existing XQuery engine results in a quick implementation for conducting the experiments.

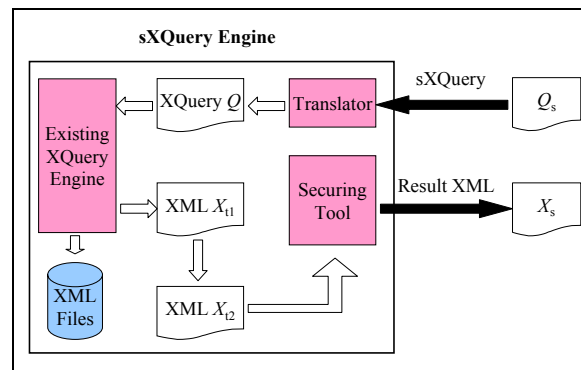


Fig. 4. The data flow of our implementation of sXQuery engine.

Fig. 4 depicts the details of our implementation of the sXQuery engine. First, the translator parses the sXQuery document  $Q_s$  and translates it into an XQuery document  $Q$ . All the sXQuery transformation descriptors in  $Q_s$  are translated into XML elements that have a tag named “**sXQuery\_Secure**”. This transforms  $Q$  into an XQuery document.  $Q$  is

then sent to the existing XQuery engine<sup>3</sup> which generates an intermediary XML document  $X_{t1}$ . The “**sXQuery\_Secure**” elements in  $Q$  are carefully arranged so that each element that should be encrypted in the XML document  $X_{t1}$  is followed by an element named “**sXQuery\_Secure**”. Figs. 5-7 illustrate an example that demonstrates the implementation.

In the next step, we translate  $X_{t1}$  into  $X_{t2}$ . Each element that should be encrypted have a new attribute named “**sXQuery\_Secure\_Template**”, the value of which corresponds to the name of the transformation template for the encryption (see Fig. 8). Finally, the securing tool parses  $X_{t2}$  to encrypt elements according to the security information described in the values of the “**sXQuery\_Secure\_Template**” attributes.

```
<transactions>
{
  for $b in doc("example.xml")/transactions/transaction
  where $b/payor = "tony yao"
  return
    $b/payor &sXQuery_Secure{TPayor}
}
<emp empid="12345">
  <name>John Smith</name>
  <job>Accountant</job> &sXQuery_Secure{TJob}
</emp>
</transactions>
```

Fig. 5. An sXQuery code ( $Q_s$  in Fig. 4).

```
<transactions>
{
  for $b in doc("example.xml")/transactions/transaction
  where $b/payor = "tony yao"
  return
    $b/payor , <sXQuery_Secure template="TPayor"/>
}
<emp empid="12345">
  <name>John Smith</name>
  <job>Accountant</job> , <sXQuery_Secure template="TJob"/>
</emp>
</transactions>
```

Fig. 6. An XQuery code ( $Q$  in Fig. 4).

```
<transactions>
<payor id="m123456789" sex="M">
  tony yao
</payor> <sXQuery_Secure template="TPayor"/>
<payor id="m987654321" sex="M">
  tony yao
</payor> <sXQuery_Secure template="TPayor"/>
<emp empid="12345">
  <name>John Smith</name>
  <job >Accountant</job> <sXQuery_Secure template="TJob"/>
</emp>
</transactions>
```

Fig. 7. An intermediary XML document ( $X_{t1}$  in Fig. 4).

```
<transactions>
<payor id="m123456789" sex="M" sXQuery_Secure_Template="TPayor">
  tony yao
</payor>
<payor id="m987654321" sex="M" sXQuery_Secure_Template="TPayor">
  tony yao
</payor>
<emp empid="12345">
  <name sXQuery_Secure_Template="TJob">John Smith</name>
  <job >Accountant</job>
</emp>
</transactions>
```

Fig. 8. An intermediary XML document ( $X_{t2}$  in Fig. 4).

<sup>3</sup> We use the XQEngine [19] as the XQuery engine.

## 5. EXPERIMENTS

We conducted experiments on the two security models shown in Fig. 2. The two-step encryption technique uses the existing XQuery implementation and a DSL securing tool (that can be found in [11]) to secure the XML document. The implementation of the sXQuery engine is described in section 4.

The resulting XML document queried from the query engine had 101 elements: a tree with one root node and its 100 child element nodes, in which each child node was associated with a text node which in turn comprised either 100 or 500 bytes. Since the encryption is performed on some of the child elements of the root element, the number of elements involved in template matching is always 101. Table 1 lists the times required for the two implementations to produce encrypted XML documents. All the experiments were performed on a PC with a 2.4-GHz Pentium 4 processor, 256 MB of RAM, the MS Windows 2000 operating system, and Java Development Kit 1.4 [22]. As shown in Fig. 4, we use the existing implementation of an XQuery engine, XQEngine, to execute sXQuery programs. Two intermediary XML files must be generated. This makes the performance of our implementation worse than that of the two-step process. In case we can integrate the XQuery engine and DSL securing tool into a single software component, we can avoid the unnecessary generation and parsing of the intermediary XML files.

**Table 1. The times required to get encrypted data using the implementation of two-step process in Fig. 2 (a) and sXQuery in section 4.**

Total elements in XML file	Number of XSL templates	Number of elements on which template matching is performed	Number of encrypted elements	Average time (in seconds)			
				100 bytes*		500 bytes*	
				Two-Step	sXQuery	Two-Step	sXQuery
101	10	101	0	0.1364	0.0560	0.1291	0.0780
101	10	101	10	0.5406	0.6906	0.4667	0.7051
101	10	101	20	0.8135	1.1219	0.7865	1.1739
101	10	101	30	0.9739	1.4687	1.1083	1.6270
101	10	101	40	1.2614	1.8114	1.3687	1.9937
101	10	101	50	1.5062	2.1562	1.6124	2.3812
101	10	101	60	1.6593	2.4572	1.8926	2.7874
101	10	101	70	1.9197	2.7156	2.0666	3.1363
101	10	101	80	2.0333	2.9615	2.2823	3.3333
101	10	101	90	2.0469	3.1562	2.4593	3.6583
101	10	101	100	2.0958	3.2937	2.6249	3.8646

\* Number of bytes to be encrypted in an element (Run-time without key and algorithm download.)

## 6. CONCLUSION

We have presented an extension to the XQuery language proposed by the W3C. This new language, called sXQuery, combines the specification ability of the XQuery language and the DSL, thereby enabling the scope and encryption details to be embedded within a single sXQuery document. We have also provided an example of an sXQuery

editor that enables users to generate sXQuery documents without having to write sXQuery source codes directly. The implementation demonstrates that the feasibility of sXQuery.

## REFERENCES

1. B. K. Schmidt, M. S. Lam, and J. D. Northcutt, "Extensible markup language (XML)," <http://www.w3.org/XML/>.
2. ISO (International Organization for Standardization), ISO 8879:1986(E), "Information processing – Text and office systems – Standard generalized markup language (SGML)," 1st ed., 1986-10-15. [Geneva]: International Organization for Standardization, 1986.
3. S. Boag, D. Chamberlin, M. F. Fernandez, D. Florescu, J. Robie, and J. Simeon, "XQuery 1.0: an XML query language, W3C candidate recommendation 3 November 2005," <http://www.w3.org/TR/xquery/>.
4. B. Schneier, *The Applied Cryptography: Protocols, Algorithms, and Source Code in C*, 2nd ed., John Wiley & Sons, New York, 1995.
5. R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Communications of the ACM*, Vol. 21, 1978, pp. 122-126.
6. National Institute of Standards and Technology, Data Encryption Standard, Federal Information Processing Standard, FIPS PUB 46-2, 1993.
7. H. Maruyama and T. Imamura, "Element-wise XML encryption," <http://www.alpha-works.ibm.com/tech/xmlsecuritysuite>, 2000.
8. P. Brandt and F. Bonte, "Towards secure XML," [http://lists.w3.org/Archives/Public/xml-encryption/2000Oct/att-0016/02-Discussion\\_paper\\_sXML.doc](http://lists.w3.org/Archives/Public/xml-encryption/2000Oct/att-0016/02-Discussion_paper_sXML.doc).
9. G. H. Hwang and T. K. Chang, "Document security language (DSL) and an efficient automatic securing tool for XML documents," in *Proceedings of International Conference on Internet Computing*, 2001, pp. 393-399.
10. G. H. Hwang and T. K. Chang, "Towards attribute encryption and a generalized encryption model for XML," in *Proceedings of International Conference on Internet Computing*, 2003, pp. 455-461.
11. G. H. Hwang and T. K. Chang, "An operational model and language support for securing XML documents," *Computers and Security*, Vol. 23, 2004, pp. 498-529.
12. M. Bartel, J. Boyer, B. Fox, B. LaMacchia, and E. Simon, "XML-signature syntax and processing, W3C recommendation 12 February 2002," <http://www.w3.org/TR/xmlsig-core/>, 2007.
13. M. Kudo and S. Hada. "XML document security based on provisional authorization," in *Proceedings of ACM Conference on Computer and Communication Security*, 2000, pp. 87-96.
14. AlphaWorks, "XML security suite," <http://www.alpha-works.ibm.com/tech/xmlsecuritysuite>.
15. T. Imamura, B. Dillaway, and E. Simon, "XML encryption syntax and processing. W3C recommendation 10 December 2002," <http://www.w3.org/TR/xmlenc-core/>, 2007.
16. T. K. Chang, "Design and implementation of the security model for XML docu-

- ments,” Ph.D. Thesis, National Taiwan Normal University, Taipei, Taiwan, 2006.
17. J. Clark, “XSL transformations (XSLT) version 1.0, W3C recommendation,” <http://www.w3.org/TR/xslt>, 1999.
  18. “XML path language (XPath) version 1.0 W3C recommendation 16 November 1999,” <http://www.w3.org/TR/xpath>.
  19. Project: XQEngine – XML Query Engine, <http://xqengine.sourceforge.net/>.
  20. Project: XQuench – XML Query Engine, <http://xquench.sourceforge.net/>.
  21. Xavier C. Franc’s Qizx/Open, <http://www.xfra.net/qizxopen>.
  22. Sun Microsystems, “The source for Java(TM) technology,” <http://java.sun.com>.
  23. C. N. Fischer and R. J. LeBlanc, Jr., *Crafting a Compiler with C*, The Benjamin/Cummings Publishing Company, Inc., 1991.

**Tao-Ku Chang (張道願)** is an assistant professor in the Department of Computer and Information Science at National Dong Hwa University. His research interests include XML related issues, internet security, object-oriented technologies.

**Gwan-Hwan Hwang (黃冠寰)** is an associate professor in the Department of Computer Science and Information Engineering at National Taiwan Normal University, Taiwan. He received the B.S. and M.S. degrees while in the Department of Computer Science and Information Engineering at National Chiao Tung University, in 1991 and 1993, respectively, and the Ph.D. degree while in the Department of Computer Science at National Tsing Hua University, Hsinchu, Taiwan, in 1998. His research interests include XML related issues, internet security, software testing, and workflow management system.