Research Report

XML: Special Topics

Version 1.4

By
Team 3

Eric Cole
Tony Gottlieb
Stella Konstantinou
David Ulmer

Assignment for DCS861A
Doctorate of Professional Studies in Computing

Pace University
School of Computer Science and Information Systems
March 2003
# Table of Contents

Part 1 – Overview of XML ............................................................................................................... 3
  What is XML? .......................................................................................................................... 3
  Historical Background ........................................................................................................... 4
  Basic XML Topics .................................................................................................................. 5
    The Internals of an XML Document ...................................................................................... 5
    XML Document Life Cycle ............................................................................................... 7
    DTD versus Schemas ........................................................................................................... 8
Part 2 – XML Transformations .................................................................................................... 10
  The XSLT Processing Model ................................................................................................. 10
Part 3 – XML Security .................................................................................................................. 14
  Architecture of XML Security ............................................................................................. 14
  XML-based Security Standards ............................................................................................. 15
  XML Digital Signatures ......................................................................................................... 16
  XML Encryption .................................................................................................................... 17
  XML-based Key Management ............................................................................................... 18
    (XKMS) Design Criteria ..................................................................................................... 18
    (X-KISS) & (X-KRSS) Specifications ................................................................................. 19
Part 4 – XML Demonstration ........................................................................................................ 26
References .................................................................................................................................... 29
Part 1 – Overview of XML

What is XML?

Extensible Markup Language (XML) is a standard set of rules for defining semantic tags that break a document into parts and identify the different parts of the document. Unlike HTML, which has a set number of tags and describes the format of the document, XML allows you to create the tags required to properly identify the semantics or structure of the document (hence its extensibility). It is essentially a meta-markup language or a language for defining other languages [4].

XML enables business to make available and share high volumes of information regardless of the software or hardware used to access it and, therefore, addresses the platform compatibility issues associated with data exchange between heterogeneous applications. XML allows designers to develop any arbitrary set of tags to describe meaning and hierarchical structure of data and supports specification of sophisticated data types required for efficient data interchange between different programs and systems. It defines a set of rules that allows designers to specify a unique set of element and attribute labels.

Why is XML important? XML can solve some long outstanding issues with universal data interchange. It is nonproprietary and easy to read (it is commonly stored in ASCII format) and write (you only need a text editor). Since it is nonproprietary, it is not subject to copyright or intellectual property restrictions which could fracture the nature of it being an open standard. Also, before the advent of XML, the common way of illustrating structure in data was through the use of complex logical and physical database relationships. XML is able to integrate semantics and structure into the body of an XML document. Lastly and possibly most important, XML allows for the design of domain-specific markup languages. Examples such as the
Chemical Markup Language (CML), Mathematical Markup Language (MathML), Information Content Exchange (ICE) and electronic business XML (ebXML) are just a few examples.

**Historical Background**

XML version 1.0 was published as a World Wide Web Consortium (W3C) recommendation standard in 1998 and Namespaces in 1999. The second edition of XML version 1.0 was published in 2000 and it essentially clarified several aspects of the standard. It remains version 1.0 since the standard did not change the allowed syntax of XML in any way. What was considered well-formed XML in the first edition remains well-formed in the second edition (essentially meaning they can be parsed).

XML is a true profile/subset of the Standard Generalized Markup Language (SGML). The origins of SGML (published in 1980 by ANSI) go back to the 1960’s with IBM’s development of the Generalized Markup Language (GML). Because SGML was inherently complex and difficult to use; HTML was developed for the narrow purpose of formatting data. However, the nature of the limited scope (i.e., data format) of HTML was a limitation for representing semantics and structure, hence the evolution and development of XML.

XML is designed to describe data in a simple, flexible, human-readable text format using either a Document Type Definition (DTD) or an XML Schema. Endorsed by software industry market leaders such as IBM, Sun, Microsoft, Software AG, Netscape, and SAP, XML is becoming the open, international standard for exchanging, storing and publishing electronic information across the IT industry. XML continues to evolve and it is also growing in complexity by evidence for its support of several RPC protocols and parsing APIs, SOAP, XLinks/XPointers and XHTML to name a few areas where the specification has expanded.

The XML standard defined ten design goals. They are [9]:
1. XML shall be straightforwardly usable over the Internet;
2. XML shall support a wide variety of applications;
3. XML shall be compatible with SGML;
4. It shall be easy to write programs which process XML documents;
5. The number of optional features in XML is to be kept to the absolute minimum, ideally zero;
6. XML documents should be human-legible and reasonably clear;
7. The XML design should be prepared quickly;
8. The design of XML shall be formal and concise;
9. XML documents shall be easy to create;
10. Terseness in XML markup is of minimal importance.

**Basic XML Topics**

The Internals of an XML Document

Listed below (table 1) is an example of an XML document that describes baseball statistics. The first line of any XML file should be the XML declaration. The declaration consists of the version attribute, which is a name-value pair structure. It is best practice to include it, although for backward compatibility, it is not required. Version 1.0 is the current version of XML so this attribute is in place for the purpose of future versions. Like HTML, tags that begin with <! represent embedded comments, which can be used to document the XML document (a DTD or schema is of course a much more effective method of “documenting” the data structure). The very first start tag is referred to as the root element (in this case <SEASON>). What follows are a series of start tags, data and/or parent/child structures and end tags. The definition of these tags may be in an associated DTD or schema.
The example above represents the statistics for a baseball pitcher. This same document can contain a separate set of tags for a non-pitching position. This XML document is a rather simple example, but of course the simplicity of XML is what makes it such an elegant solution for data interchange. This example also illustrates some of the advantages of XML such as [4]:

1. The data is self-describing;
2. The data can be manipulated with standard tools (e.g., text editor);
3. The data can be viewed with standard tools (e.g., browser);
4. Different views of the same data are easy to create with style sheets;
5. Support of double-byte Unicode.

**XML Document Life Cycle**

The life cycle of an XML document depends on whether it is bound for direct use by a human, say in a browser window or a database. To be compliant XML documents, they must conform to the XML standard, meaning they must be well-formed and valid. A text document is a well-formed XML document if [9]:

1. Taken as a whole, it matches the production labeled document;
2. It meets all the well-formedness constraints given in the specification;
3. Each of the parsed entities which are referenced directly or indirectly within the document is well-formed.

Validity is the concept of comparing an individual XML document to its associated DTD and finding that it matches all elements described in the DTD. Likewise, if a mismatch is found, the document is considered invalid (although it may still be well-formed) [9].

The life cycle of an XML document is illustrated in figure 1. Each of these phases are decoupled and independent processes only connected by the actual XML document [4].

![Figure 1: XML Document Life Cycle](image-url)
The first phase of the life cycle is document creation, which can be achieved with a text editor or through programmatic means resulting in an XML document. The second phase is parsing the XML, which is the process of an XML processor (often called an XML parser) reading the document and verifying that it is well-formed. If the document is accompanied by a DTD (which is optional), it can also be checked for validity by the parser. If the parser can successfully read the document, it converts the document into a tree of elements. The third phase can be performed by a variety of different programs ranging from a browser (if the intent is to view the XML data structure) or a program (in the case of the XML document being written to a database).

**DTD versus Schemas**

DTD became a popular validation and data transfer format for SGML a long time before XML was developed. Therefore, it does not comply with the formal specifications of XML document rules. The purpose of DTDs is to define the basic requirements on the structure of an XML document (e.g. describe the elements and attributes valid in an XML document and the context within which they are valid). Currently, the majority of technological advances and all new W3C specification are focusing on XML Schemas rather than DTDs. As a result, future DTD vendor support is expected to diminish considerably.

The emergence of XML Schemas was inspired by the need to define strong data typing of element content and attributes, integrate validation into XML infrastructure, use XML syntax to describe grammar and integrate namespaces into grammars. The release of XML Schemas provided a more powerful mechanism of XML document validation by allowing machines to execute user specified rules and express shared models of business vocabularies which provide a means for defining the structure, content and semantics of XML [11]. XML Schemas are
becoming more popular than DTDs in the IT industry because they adapt formal specifications of the rules of an XML document.

The W3C XML Schema (WXS), a W3C Recommendation, was approved on May 2, 2001 with an intention to address many of the limitations of DTD. The following list describes the most popular features included in XML Schemas:

- Built in mechanisms for constraining document structures and content
- Ability to embed documentation
- Constraints based on document class
- Provision of direct support for extensions
- Support of keys and referential integrity constraints
- Ability to validate documents composed from markup from multiple namespaces resulting in sharing of data by multiple organizations
- Integration with primitive data types
- Mechanisms enabling inheritance for element, attribute, and data type definitions to formally represent kind-of (for example, an oak is a kind-of tree) and part-of relationships.
Part 2 – XML Transformations

Cascading Style Sheets do not have the capability of performing specialized operations such as sorting, searching or combining data from multiple sources. To accomplish these tasks required when dealing with structured information technologies, the World Wide Web Consortium (W3C) working group responsible for stylesheets defined the Extensible Stylesheet Language Transformations (XSLT) language and published it as a recommendation on November 16, 1999. Additionally, the above group worked in partnership with the W3C working group responsible for the next generation of hyperlinking to produce the XML Path Language (XPath), a common syntax and semantics used by XSLT to address parts of an XML document and associate them with other information [10].

XSLT enables and empowers interoperability [6]. It uses XML vocabularies as its target to transform XML documents into other text formats such as HTML, WML, XML, PDF, etc. or a different XML document that conforms to another DTD. An XSLT Stylesheet describes rules for transforming a source tree (or input tree) into a separate result tree. “The transformation is achieved by associating patterns with templates. A pattern is matched against elements in the source tree. A template is instantiated to create part of the result tree. The structure of the result tree can be completely different from the structure of the source tree. In constructing the result tree, elements from the source tree can be filtered and reordered, and arbitrary structure can be added.” [12]

The XSLT Processing Model

The following XML document is a partial menu list used in our sample application.

```xml
<?xml version="1.0" encoding="UTF-8" ?>
< MENULIST>
  < ITEM>
    < NAME>Fries</ NAME>
  </ ITEM>
</MENULIST>
```
In order to perform a transformation, the XSLT processor parses the source document to produce a tree representation based on the document contents and structures. Each generated node includes the node type, node name and either the node text content or, in the absence of text content, an asterisk. The top level of the tree is the document root. The second level includes the document root’s children (for example, document element, XML declaration, processing instructions, etc.). In our example, <MENULIST> is the document element which has two <ITEM> children. <NAME>, <DETAILS>, <GRAPHICS> and <TYPE> are each <ITEM>’s
children. Finally, <CATALOGID>, <DESCRIPTION>, <SIZE> and <PRICE> are each <DETAILS>’s children.

XSLT relies on a parser to convert the XML document into a tree structure. There are two major parser types, the Simple API for XML (SAX), an event-based interface where the application is notified by the parser of individual document information while the document is read and the Document Object Model (DOM) parser, a platform and language independent interface that allows dynamic access and update of the content, structure and style of documents [8]. The DOM parser interrogates the document using a custom written application and builds a tree-based structure in memory.

XSLT uses XPath expressions to match nodes when transforming an XML document into a different document. The XSL processor will apply template rules to the nodes (including nested nodes) matching the pattern specified by the template. In our example,

\[
\text{<xsl:template match="/MENULIST"> ... </xsl:template> will match the root element \( <MENULIST> \) and all its descendants. The second template <xsl:template match="NAME">...<</xsl:template> will match \( <NAME> \) which will be displayed in ascending order by the <xsl:sort select="NAME" order="ascending" /> ...</xsl:sort> element. When the XSL processor encounters the first template it will start reading the root element. The XSL processor will use the second template to match and display detailed \( <ITEM> \) information for each \( <NAME> \).}

Whenever the XSL processor encounters a for-each element it performs a recursive loop over every instance of each matching selection. In our example, when the processor encounters \text{<xsl:for-each select="ITEM">} it will loop over the two instances of \( <ITEM> \) and it will extract the values of the selected nodes using the value-of element (for example, \text{<xsl:value-of select="NAME" />} will extract the \( <NAME> \) value).
Whenever the XSL processor encounters `<xsl:element>` `<xsl:attribute>` it will create an element and an attribute respectively with the specified name in the output. The XSL processor uses the following three elements together for conditional case testing as demonstrated by the code below:

![Example code snippet]

Table 3: XSL Processor

- `<xsl:choose>` to set up the template.
- `<xsl:when>` to specify case conditions which if tested true, will have the associated XSL rules applied.
- `<xsl:otherwise>` to set up the rules to apply when none of the `<xsl:when>` cases are true.

The above elements are essential elements in writing XSLT stylesheets to display data. However, there are many more XSL elements that provide flexibility and control over data, which can be transformed and re-used in a variety of formats and contexts.
Part 3 – XML Security

Whenever you deal with a new way to represent or transmit data it is critical that security is accounted for. If it is not accounted for, sensitive data can be compromised and the company could lose large amounts of information. Therefore it is critical that security is addressed in any new technology on two fronts:

1) make sure the technology itself is not vulnerable and cannot be compromised
2) make sure how the technology handles the data is secure

Unfortunately the way it works with most new technologies is that little attention is paid to the first area and most of the time is spent on the second area. That is also true of XML. XML does make some security assertions, but a bulk of the effort in XML is to protect the data through various means of encryption. Encryption can be used for protection of confidentiality and also for digital signatures and access control.

Architecture of XML Security

In order to make sure security is properly addressed XML provides a list of technical standards that are incorporated into the XML language. The standards make it easy to be re-used across applications and allow security to be easily integrated into existing and new XML applications. The XML security standards leverage existing standards and technology and are described as follows [5]:

- The XML language has specific tags reserved for dealing with security related information. One of the more commonly used tags is the <KeyInfo> tag which is used for signing or encrypting information.
The XML security standards use current XML tags whenever possible to try and make the information as secure as possible. An example of a normal tag is XMLDigSig and is extended to incorporate security in the XpathFilter tag.

The XML security standards can be applied to any part of the XML language. This includes documents, elements and element content.

XML security can be provided end-to-end in transit. This type of security is very similar to the security provided by SSL and TLS.

XML security builds on existing measures whenever possible. For example for digital certificates it does not reinvent the wheel, it uses X.509 certificates. Other hash algorithms like SHA are also incorporated directly into XML.

**XML-based Security Standards**

The core security standards for XML are:

- XML uses digital signatures for integrity, non-repudiation and signatures. This is done through using hash algorithms with public private key pairs.
- XML encryption for confidentiality. This is achieved through using various forms of symmetric key encryption.
- XML key management (XKMS) which is done through encryption and X.509 certificates.
- SAML or the security assertion markup language used for authorization and authentication.
- XACML or the XML access control markup language for adding in authorization rules.
XML Digital Signatures

As with any technology including XML, digital signatures are used to prove that the message came from a specific party. This is often referred to as non-repudiation and allows a party to prove in a court of law than someone sent a give piece of data. It is also used to prove that the message was not modified in transit or subject to an integrity attack.

Digital signatures are also tied directly to a message, which means if the message changes in any way shape or form the digital signature will not be valid. This is more robust than simple integrity schemes like parity checks where it is fairly easy for someone to bypass. Since digital signatures are unique for a given item they cannot be moved from XML data structure to a different data structure without being re-created.

With the XML digital signature you can digital sign the following items:

- Entire XML documents including specific elements within the document. What is also interesting is that different elements within the document can be signed by different people and than the entire document signed by a completely different person. This gives the maximum amount of flexibility where different people can update various portions and still verify the integrity of the entire document.
- Any XML format including straight ASCII or binary files.
- Multiple documents or multiple elements across documents. Since one person might be responsible for several documents or for certain elements with various documents, the digital signature can be applied across many files or data structures.
- Allows XML components or documents to be counter-signed by multiple entities to ensure the validity across many parties.
A signature can be directly embedded within an XML data structure or it can be stored as a separate data structure. This allows for the data to be transmitted and the signature verified at a different place. The following shows a signature directly embedded within an XML data structure:

```xml
<CustomerRecord xmlns="http://www.somestore.org/"
    xmlns='http://www.w3.org/2001/11/xmldsig#'>
    <Name> Frank Smith </Name>
    <Account> 555345 </Account>
    <Visit date="11pm April 14, 2002">
        <Order> Ordered 10 widgets </Order>
    </Visit>
    <Signature xmlns='http://www.w3.org/2001/11/xmldsig#'>
        ...
    </Signature>
</CustomerRecord>
```

Table 4: XML Signature Document

The signature can also be in a separate file as shown:

```xml
<Signature xmlns='http://www.w3.org/2001/11/xmldsig#'>
    <SignedInfo> ... </SignedInfo>
    <SignatureValue> ... </SignatureValue>
    <Object>
        <SignatureProperties>
        </SignatureProperties>
    </Object>
</Signature>
```

Table 5: XML Signature Document

**XML Encryption**

XML provides two types of encryption. One is for data stored on a hard drive and one is for data that is in transit. The process used for encrypting data in XML is similar to most other applications:

1. a symmetric key is generated and used to encrypt the data
2. the symmetric key is than encrypted with the public key of a recipient
3. the two pieces of encryption are put together and either stored on the system or sent across the wire

The `<EncryptedData>` tag is used to encrypt data within a data structure. The following shows an XML element with the encryption added in:

```xml
<CustomerRecord xmlns="http://www.somestore.org/">
    <Name> Frank Smith</Name>
    <Account> 555345 </Account>
    <Visit date="11pm April 14, 2002">
        <Order> Ordered 10 widgets </Order>
    </Visit>

    <EncryptedData Type='element'>
        ...
    </EncryptedData>

    <Signature xmlns='http://www.w3.org/2001/11/xmldsig#'>
        ...
    </Signature>
</CustomerRecord>
```

<table>
<thead>
<tr>
<th>Table 6: XML Encryption</th>
</tr>
</thead>
</table>

**XML-based Key Management**

Key to the XML Trust Model is the relationship between the XML Signature and Encryption standards and the XML Key Management Specification. Where the XML Signature and Encryption standards conduct all PKI related business, the XML Key Management Specification (XKMS) provides the programming interface for web developers to such services.

**(XKMS) Design Criteria**

The design criteria for XKMS are:

1. To be compatible with the XML-Signature and XML-Encryption standards;
2. Implementation should be simple (*All Cryptographic actions stay within XML Signature & Encryption*);
3. ASN.1 tools are not required (*Data Definition for Telecommunications*);
4. Management of status information is transparent to a public-key using application in an on-line environment. Here, an application need not query the active status of a key. The application simply asks for a key to perform some service and if there is a key for that service, the key will be returned. If the key has been revoked, a key will not be returned;

5. Minimize client code and configuration complexity. (*Providing standardized protocols and interfaces*).

**(X-KISS) & (X-KRSS) Specifications**

The XKMS related services include the distribution and registration of public keys. XKMS delegates the distribution of public key services to its XML Key Information Service Specification (X-KISS) and further delegates the public key registration services to its XML-Key Registration Service Specification (X-KRSS).

**(X-KISS)**

The XML-Key Information Service Specification (X-KISS) fetches information related to a public key. These pieces of information are referred to by *KeyInfo elements* and they are used to authenticate a digital signature associated with an incoming message to the application. The signor of the message may or may not attach KeyInfo elements such as their public key to the message. In the case where the public key (*KeyValue*) is attached, key verification:

- Begins locally, by the receiving application, with the signatory’s public key being used to decode the footprint in *SignatureValue* followed by comparison to the locally obtained footprint. This comparison will detect any changes to the document during transfer;
- Is followed by (X-KISS) *Locate & Validate* service requests for additional information regarding the received public key or a status check on the key’s validity or both.
According to the XML Signature Specification, the signor may elect to attach information about their public key. This information may include one or a combination of the following KeyInfo elements:

1. “Hints” to those who wish to verify a key; as to which of the signor’s keys to choose;
2. A RetrievalMethod (A referral for verifiers via URL, to additional KeyInfo information held remotely);
3. KeyName (The name of the signors key);
4. X509Data (The signor’s certificate written in std. X509 format).

The key information referred to above are maintained by a Trust Service. The X-KISS protocol permits applications (clients) to delegate processing of XML-Signature elements to a Trust Service. Architecturally, the Trust Service can be located on a different server than the client (application) and all messaging between them employs the XML format. The managed PKI services, which return key information requested by the Trust Service may itself be located on its own server. Given the X-KISS protocol architecture, there are two key information retrieval services available to applications (clients). They are the Locate service and the Validate service.

(X-KISS) Locate Service

The Locate service is called upon by applications to retrieve additional key information which has been inserted into KeyInfo. A typical Locate service request would be the retrieval of a public key from the signatory’s name. Another example would be identifying the owner of a public key. The architectural placement of the KeyInfo data requested may be local to the Trust Service or on a separate server and the communication from The Trust Service to this underlying PKI server may utilize a non-XML syntax; such as HTTP. It is important to understand the service request originates with the application (on server 1) requesting service from the Trust
service (on server 2) and communication between them is in XML format. The Trust service may then pass the request on to the PKI service (on server 3) for final retrieval of the KeyInfo element requested.

Below, we see a Locate service request by an application (client) who has received a XML document which has been signed. The retrieved KeyInfo element provides for a (URL RetrievalMethod) for an X509 certificate which contains the public key.

```
<Locate>
  <Query>
    <ds:KeyInfo>
      <ds:RetrievalMethod
        URL=http://www.somebody.test/Certificates/02191366
        Type=http://www.w3.org/2001/09/xmldsig#X509Data/>
    <ds:KeyInfo>
      <Query>
        <Respond>
          <string>KeyName</string>
          <string>KeyValue</string>
        </Respond>
      </Query>
    </Locate>

Table 7: Locate Service Request
```

Observe the code inside the <ds:KeyInfo> block which does indeed specify and URL and Type components of the RetrievalMethod KeyInfo element. Here, the URL specifies where the certificate can be found and the Type specifies the data type as X509. Notice too, the <Respond> block, which contains the KeyInfo elements (variables), KeyName and KeyValue. When this request is serviced, either at the Trust service (server 2) or at a separate PKI server (server 3), the requesting application will be able to access the KeyName and KeyValue KeyInfo elements returned by the PKI service.

**(X-KISS) Validate Service**

The Validate service is called upon by applications (clients) to assert that the binding between a public key and a data element is in good standing. As more than one data element may be bound to the same public key, the validate service will assert the validity of each binding.
Procedurally, an application (client) will pass to a Trust service, the elements whose trust binding status is desired. If the Trust service receives only partial information needed to satisfy the request, the Trust service will then request any additional data from the PKI service; which may reside on a separate server. The Trust service will validate the key binding after receiving the requested data from the PKI service then pass the result back to the application (client), where the request originated.

Below is a fragment of code from an application (client), destined for the Trust service. It shows a verified document’s binding being queried for its trustworthiness and its validity.

```
<Validate>
  <Query>
    <Status>Valid</Status>
    <ds:KeyInfo>
      <ds:KeyName>…</ds:KeyName>
      <ds:KeyValue>…</ds:KeyValue>
    </ds:KeyInfo>
  </Query>
  <Respond>
    <string>KeyName</string>
    <string>KeyValue</string>
  </Respond>
</Validate>
```

Table 8: Trust Service

Notice the label in the outer-most block (e.g., Validate), thus directing the request to the validate service that is part of (X-KISS). Notice too, the values for the KeyInfo elements KeyName and KeyValue being passed to the Trust service which will provide a basis for comparison between what the application knows as KeyName and KeyValue and what the PKI service knows as KeyName and KeyValue. The response from the Trust service to the application appears below:

```
<ValidateResult>
  <Result>Success</Result>
  <Answer>
    <KeyBinding>
      <Status>Valid</Status>
      <KeyID>http://www.xmltrustcenter.org/assert/10101020-99</KeyID>
      <ds:KeyInfo>
```

22
Table 9: Response from Trust Service

One important aspect of the above response is the ability for the application (client) to access, store and manipulate the returned data it considers valuable. For example, consider the `KeyInfo & ValidityInterval` Blocks. Here is where the results of the comparison between `public key/name` and `time interval` for the keys validity are returned.

Applications making requests of the Locate or Validate services should be aware of the risk associated with invalid responses. For a response to be valid, it must have the following attributes:

- Authenticity: Response returned from a trusted Trust service;
- Integrity: Response returned to application (client) has not been garbled in transit;
- Correspondence: Response returned is positively linked to the correct request.

The above secure attributes may all be satisfied by the following secure mechanisms:

- Authentication by XML Signature;
- TCP/IP Transport Layer Security (e.g. SSI, TLS, WTLS);
- Packet Layer Security (e.g. IPSEC).

(X-KRSS)

The XML Key Registration Service Specification is called upon by clients to:

- Register Key  (Registration servers may request this too);
- Revoke Key;
- Recover Key.

In each instance, authentication is achieved by the user providing a password (`PassPhrase`).

Registering-
Registering is the generation of a public key pair that is bound to information. This “binding” is represented by and referred to an XKMS <KeyBinding> element. The request for such a binding is called an assertion and is aimed at the registration service. Upon receipt of the requested assertion, the registration service may:

1. Request additional client information for authentication;
2. When request originates from client, *Proof-Of-Possession* of a private key may be requested;
3. Find everything having to do with the assertion in order and register the assertion.

Registration requests originating from clients differ from those originating from a service. A service based assertion will not have included in its prototype, the public key data. The request will also dictate the returning of the private key data. This is because a server-side generation is creating a public/private key pair. The client therefore doesn’t know either, until the server responds.

Revoking-

Here, a client may revoke an assertion and thus render it invalid. One who has been authenticated by signing their revocation request with the private key associated with the public key who’s binding (assertion) is to be revoked, could have provided the *PassPhrase* returned during registration as an alternative way to authenticate. It is important to understand key revocation as an intrinsic part of PKI. No key will last indefinitely and it is therefore recommended that a key revocation certificate be generated at the time the key pair is created.

Recovery-

A forgotten PassPhrase for a private key is a situation to be avoided. The consequences of such a problem include unreadable mail that has been encrypted to that key, the inability to sign outgoing mail, the inability to create a key revocation certificate and thus the inability to
give notice that the associated public key is now ineffective. Certainly, the forgetting of a PassPhrase renders the key pair unsuitable for the purpose for which it was intended.

Two approaches to PassPhrase recovery:

First, a *key-escrow facility* can hold a copy of the owner’s private key that has been serially encrypted in the public keys of that *key certification authority*. Here, the owner may regain the use of the forgotten private key by first providing strong identification.

Second, an authorization code could be issued to the client by the certification authority, which will revoke the forgotten private key when the client passes the authorization code back to the service via the application.
Part 4 – XML Demonstration

The XML application demonstration is an example of a XML Transformation using the Extensible Stylesheet Language (XSL). In this demonstration XML is transformed into a different form of text, in this case Extensible HyperText Markup Language (XHTML). The XML processor is a Java program (Transform.Java) [2] that uses the JAXP library’s XSLT processor to perform the XSL transformation.

The XML, which represents a menu (table 10), is transformed with the XSL (table 11 – sample only) into the XHTML page (figure 2).

```xml
<?xml version="1.0" encoding = "UTF-8"?>
<MENULIST>
  <ITEM>
    <NAME>Fries</NAME>
    <DETAILS>
      <CATALOGID>1</CATALOGID>
      <DESCRIPTION>Fries</DESCRIPTION>
      <PRICE>$1.00</PRICE>
      <SIZE>Large</SIZE>
    </DETAILS>
    <DETAILS>
      <CATALOGID>2</CATALOGID>
      <DESCRIPTION>Fries</DESCRIPTION>
      <PRICE>$1.50</PRICE>
      <SIZE>Medium</SIZE>
    </DETAILS>
    <DETAILS>
      <CATALOGID>3</CATALOGID>
      <DESCRIPTION>Fries</DESCRIPTION>
      <PRICE>$0.90</PRICE>
      <SIZE>Small</SIZE>
    </DETAILS>
    <GRAPHIC>./Images/fries.gif</GRAPHIC>
    <TYPE>
      <SO>Special Order</SO>
    </TYPE>
  </ITEM>
</MENULIST>
```

Table 10: XML of Menu items
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns = "http://www.w3.org/1999/xhtml" version="1.0">
    <xsl:output method="html" omit-xml-declaration="no" doctype-system = "-//W3C//DTD XHTML 1.0 Strict//EN" />
    <xsl:param name="which" select="'INDEX'"/>
    <html>
        <body>
    </body>
    </html>
    <xsl:template match="/MENULIST">
        <xsl:for-each select="ITEM">
            <xsl:sort select="NAME" order="ascending"/>
            <TABLE ALIGN="CENTER" BORDER="0" cellpadding="6" cellspacing="6"
                WIDTH="700">
                <TR>
                    <TD WIDTH="50"></TD>
                    <TD VALIGN="MIDDLE" HEIGHT="100">
                        <form NAME="details">
                            <A HREF="report.html" onclick="alert(nm.value);">
                                <xsl:value-of select="NAME"/>
                            </A>
                            <INPUT NAME="nm" type="hidden">
                            <xsl:attribute name="VALUE">
                                <xsl:value-of select="NAME"/>
                            </xsl:attribute>
                        </form>
                        <TD ID="graph" VALIGN="MIDDLE" HEIGHT="100">
                            <xsl:choose>
                                <xsl:when test="GRAPHIC[. !='']">
                                    <INPUT TYPE="IMAGE" NAME="pict">
                                    <xsl:attribute name="SRC">
                                        <xsl:value-of select="GRAPHIC"/>
                                    </xsl:attribute>
                                </xsl:when>
                            </xsl:choose>
                        </TD>
                    </TD>
                </TR>
            </TABLE>
        </xsl:for-each>
    </xsl:template>
</xsl:stylesheet>

Table 11: XSL for Menu items XML
Figure 2: XHTML from XSL transformation

<table>
<thead>
<tr>
<th>Item Description</th>
<th>Size</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>All you can eat chicken - Served weekdays from 9:00 A.M to 10:00 P.M</td>
<td>N/A</td>
<td>$5.00</td>
</tr>
<tr>
<td>All you can eat pork - Served weekdays from 9:00 A.M to 10:00 P.M</td>
<td>N/A</td>
<td>$5.00</td>
</tr>
<tr>
<td>All you can eat chicken and pork - Served weekdays from 9:00 A.M to 10:00 P.M</td>
<td>N/A</td>
<td>$5.00</td>
</tr>
<tr>
<td>Bagel with butter</td>
<td>1 bagel</td>
<td>$0.50</td>
</tr>
</tbody>
</table>
References


