http://ansinet.com/itj

ITJ

ISSN 1812-5638

INFORMATION TECHNOLOGY JOURNAL

ANSInet
Asian Network for Scientific Information
308 Lasani Town, Sargodha Road, Faisalabad - Pakistan

Information Technology Journal 6 (7): 996-1004, 2007
ISSN 1812-5638

XML Context's Security Patterns Language: Description and Syntax

Tawfiq S.M. Barhoom and Zhang Shen-Sheng
Computer Integrated Technology Lab, Shanghai Jiao Tong University, Shanghai, 200030, China

Abstract: The recently increasing adopting of XML as base technology for web application, increasing the need to make its security available and understandable. At present, it can be observed that weakness of security of web application may be because developers are lack of security knowledge and the security is not considered as a primary function. Patterns plainly capture the experience from experts in a structured way. Thus, a trial is made to apply the pattern approach to the XML security context, presenting a security patterns language in XML security context. The pattern language has three abstraction levels. Its form based on the user's level of understanding of the security and XML security fields.

Key words: Security patterns, security pattern languages, XML security, security patterns relations

INTRODUCTION

The security factor is important for the applications, so that the designers and the developers whom are not a security people should keep an eye on the security issues during the application development, in order to avoid repeated mistakes. By looking-up at the application team, there are some who became experts and others are new comers to the security, from this heterogeneous set we may wonder: How can we give them security components they may need to use allowing them to focus on their application and considering care about the security issues? and How the novice can get benefit from know-how and skills of experts? well, patterns may have the answer for all these questions. A pattern is used to describe best practices and principles in a structure way. The purpose of a pattern is to increase the productivity of the designers by making them profit from the experience gained on former projects (Appleton, 1997). Using patterns has number of significant benefits (Vlissides, 1998): Capturing expertise, Give names to higher-level constructs, Document the strengths and weaknesses of different architectural options and identify troublesome details. The field of security is very broad; we cannot speak of a pattern language that implies the complete coverage of every aspect of the problem domain (Schumacher and Roedig, 2001). Present purpose is to improve the state of cit-lab security by applying the patterns way with security problem at XML-Based application. Using this way to document and organize number of problems in these applications, this can result in making the use of this approach in reusing security expertise available. If patterns can be used to capture security expertise and present it to the developer community in a relatively painless manner, this might go a long way towards our target. This study is looking after the language description; what are the elements of the language and how organized in three levels and how the elements from different levels can be combined in a format (syntax) gives a security sentence in the domain.

PATTERNS AND SECURITY PATTERN

Patterns are useful tools for inexpert to know how experts think. Each pattern describes both a problem which occurs over and over again in our environment and then describes the core solution to that problem, in such a way that you can use this solution million times over, without ever doing it the same way twice (Alexander, 1977). In short; a pattern constitutes a solution to a problem in a context (Lea, 1997).

Security patterns are a variation on Patterns. One definition of security pattern is that a security pattern describes a particular recurring security problem that arises in specific contexts and presents a well-proven generic scheme for its solution (Schumacher and Roedig, 2001). Each security pattern will document a recurring solution to solving a general class of security problem. Each will contain among other things a description of the problem, an outline of the solution and enough rationale to help understand why it works, when it is applicable and how to apply it in novel situations (Gamma *et al*., 1995)

THE LANGUAGE OVERVIEW

A pattern language is a collection of patterns that work together to solve problems in a given domain. The patterns of our language categorized into three levels of

Corresponding Author: Tawfiq S.M. Barhoom, Computer Integrated Technology Lab, Shanghai Jiao Tong University, Shanghai, 200030, China

abstraction they are; security requirements patterns level, relevant XML-Based security patterns level and the process patterns. The first level guides to the minimum-security requirements for web application while the second level translates these requirements into XML-Based security technologies (Kreger, 2001). Finally the third level is the process patterns that give the process steps at the end of the transaction, the description of the language and its levels are shown in Fig. 1.

Patterns for different purposes usually do not exactly have the same format (template) and for each purpose an adaptation is needed. We discuss the template for XML-Based security patterns in (Tawfiq and Zhang, 2005). The security patterns template that we developed for XML-Based security patterns were derived in part from research into existing patterns templates, including ones used by AG Communication Systems and the Gang of Four in Design Patterns.

We have examined previous patterns templates and settled on the structure described at (Tawfiq and Zhang, 2005) for our security patterns. Since the patterns composed within the language are classified into three abstraction levels that to be used by different users and different purposes, therefore two sections format and solution have different descriptions for each level to meet the purpose as shown in the Table 1. We do not claim that our template is perfect or complete. We just started the project with a general idea of what should be in the XML security patterns template (Table 1).

A pattern does not stand-alone; the relationships among the security patterns are the clue to organize these patterns into a Security Pattern language. These patterns should cooperate in solving a problem according to relations amongst them, another requirement the security patterns language should meet is that, the relationships between security patterns have to be exposed (Zimmer, 1995). We find three relationships between the patterns from the three levels; these relationships are provided, simplified and process-of relationships. These descriptions of these relations as follow:

Provided relationship: A pattern for security of XML document provides one or more requirements (Fig. 2).

Simplified relationship: A pattern of XML for security function simplifies one or more requirements (Fig. 2).

Process-of relationship: Any pattern for security of XML document or pattern of XML for security function should have two processes for each end (sender and receiver). A pair process' patterns from the third level are processes of a pattern from the second level (Fig. 3).

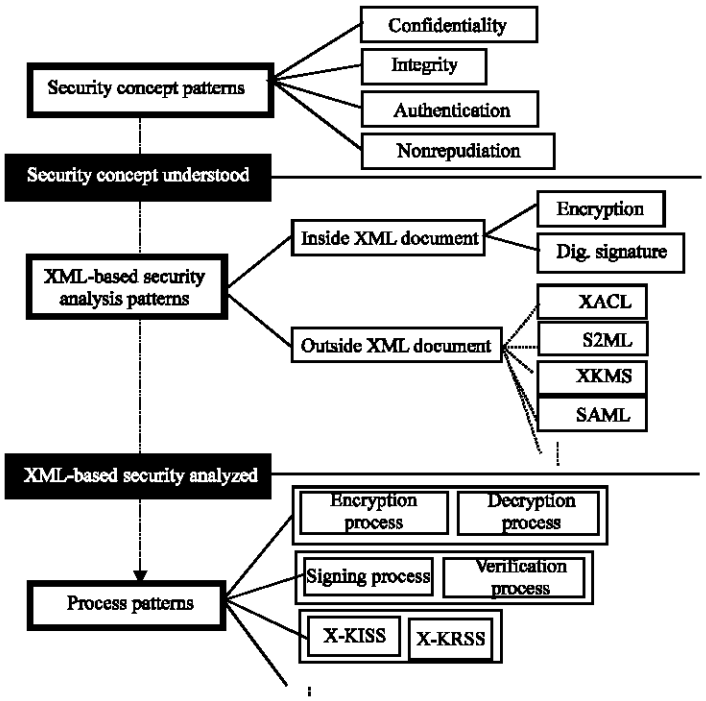


Fig. 1: Description of the XML-based security patterns language

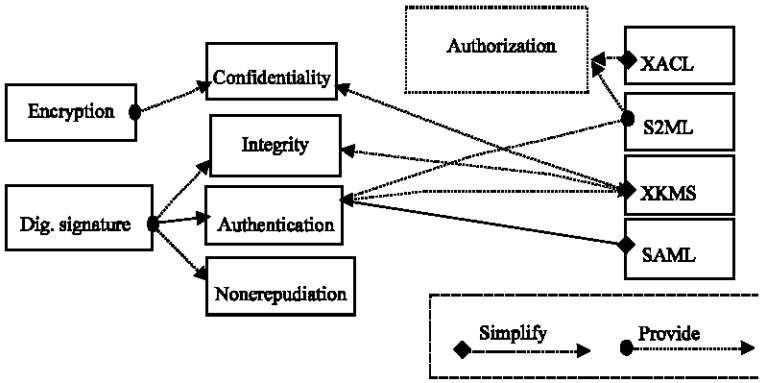


Fig. 2: XML-based security patterns related to the web application security requirements

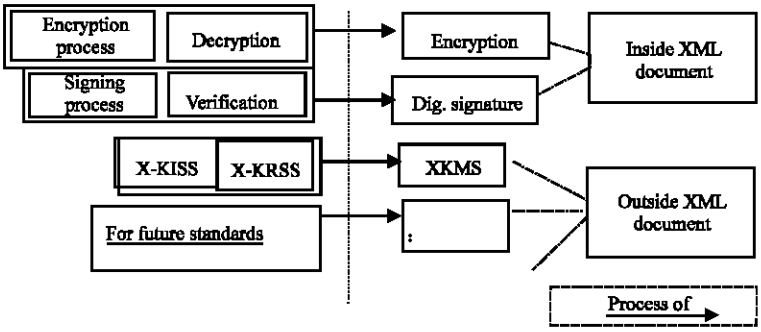


Fig. 3: A pair process' patterns from 3rd level are processes of an XML-based security

Table 1: Description of the XML-Based security patterns language

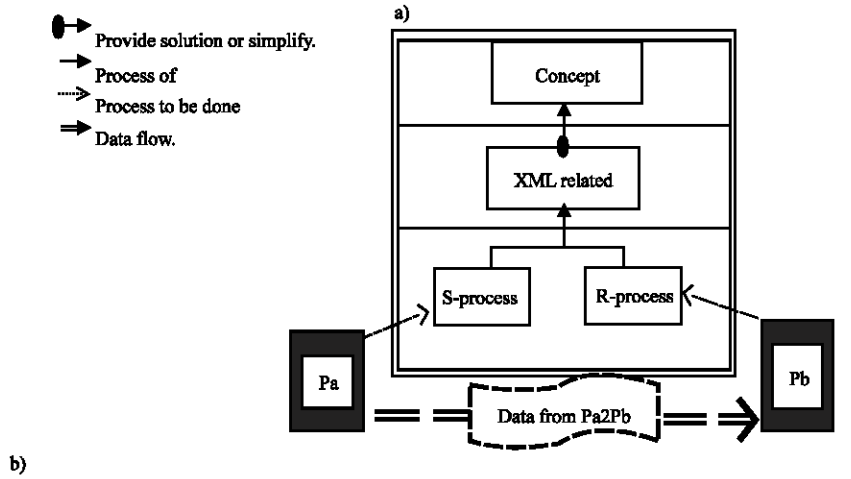
Field name	Description
Format	Concepts patterns: A diagram that illustrate the secutity concept and general steps of precess in solving the problem.
	XML analysis patterns: The physical structure of the corresponding secutity element to security data should be presented.
	Precess patterns: A diagram possibly a set of diagrams that show the flowchart of the implementations process needed.
Solution	Concepts patterns: List the general steps of the process in solving the security problem.
	XML analysis patterns: Proposes a solution to the problem that specifies a configuration of XML elements to balance the forces associated with the security summarizing the process one is following while applying the pattern. This section may include a Steps or list of Hints to be taken in account when applying the solution.
	Process patterns: List the core classes-like to implement the problem.

USAGE OF THE LANGUAGE

Using this language is user dependent since there are heterogeneous set of possible users with variety in understanding the domain; each user has a basic level depending on his/her understanding of the domain. Novices and new comers to the security go along the levels, understanding the security concepts by using the patterns of the first level through the third level. Some other users have already understood the security concepts but not XML security; for such users the second level is the starting level through the third.

Going from top to bottom through the language levels help in having a solution for the security requirements. Understanding the concept in advance is so important; while understanding how XML technology deals with the concept is required before the implementation stage takes place. This strategy can be considered as a presentation for the language's production in a format (syntax) as shown in Fig. 4, the production called a sentence. The sentence is a combination of four related patterns from the three levels that gives the user a meaningfully of security concept from concept-understanding to XML-implementation of that concept.

The top level's pattern gives the general security concept; the second level's pattern gives the concept in XML world and the last level's patterns give a guide to implement the concept within XML applications. The later



b)

```
Requirement    := CONFIDENTIALITY | INTEGRITY | AUTENTICATION |NONREPUDATION

RequirementSol  := RSimplified | RProvided

RSimplified  := ConceptP, XMLoUTP, ProcessPair

RProvided  := ConceptP, XMLiNP, ProcessPair

ConceptP      :=   Confidentiality |  Inegrity  |Authentication |Nonerepudiation

KMLiNNP       :=   Dig signature| Encryption

KMLoUTP       :=  XKMS | XZCL | S2ML | SAML  |...

ProcessPair   :=  SigningPa   EncryptPa | XKMsPe

SigningPa     :=  ("Signing process", "Verification process")

EncryptPa     :=  ("Encryption process", Decryption process") |

XKMsPa        : = ("K-KISS", K-KRSS")
```

Fig. 4: The sentence combined four patterns from the three levels of the language a) Sentence format, b) BNF; lay out the format of the Sentences

consist of tow patterns, the first on is sender process (S-process) pattern and the second is the response what we call receiver process (R-process) pattern.

SENTENCE EXAMPLE

The sentence presented in this section is one product of the language described above. The meaning of the sentence gives the user the basics of Non-repudiation concept in web application by the "Non-repudiation" pattern then presents the concept in XML technology in Signing an XML object (Fig. 5) that provides Non-repudiation for the web application and finally the process steps in both ends by two patterns, first S-process Signing process pattern and the second R-process Verifying process pattern (Fig. 7 and 8).

Security concept patterns (non-repudiation):

Forces The sender's digital signature is associated with a pair of keys: private key and public key.

Format:

Solution: Figure 6 shows a typical digital signature process:

- The data to be signed and the private key are the inputs to a hashing process.
- The output of the hashing process (data digest), encrypted with the sender's private key (digital signature).
- The data, the digital signature and the sender's public key (sender's certificate) in some case are sent to the recipient.

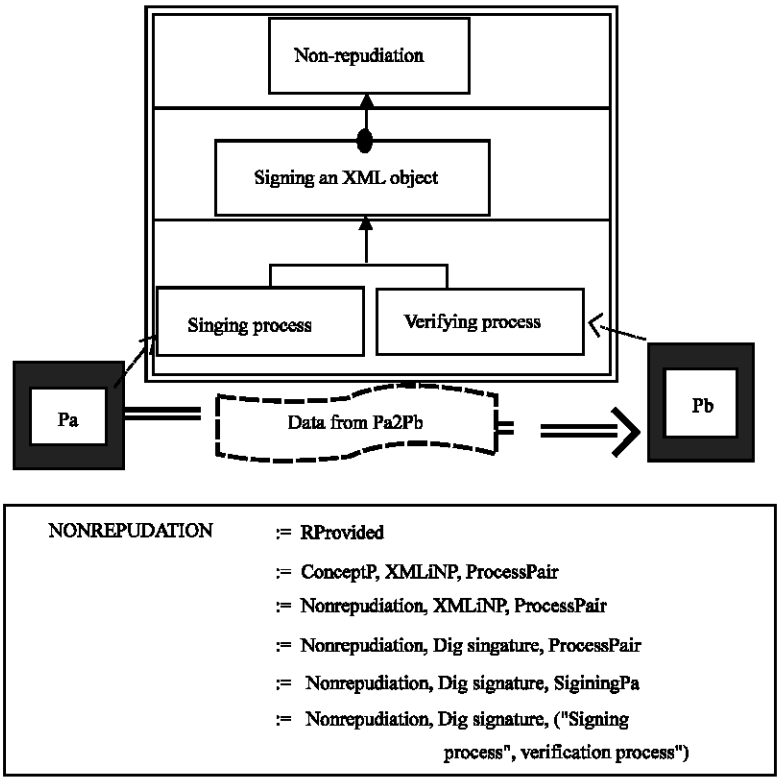


Fig. 5: Provide Non-repudiation Sentence; combined Non-repudiation patterns from security requirements patterns level, Signing an XML object pattern from XML-based security patterns level and Signing process and Verifying process from the process level

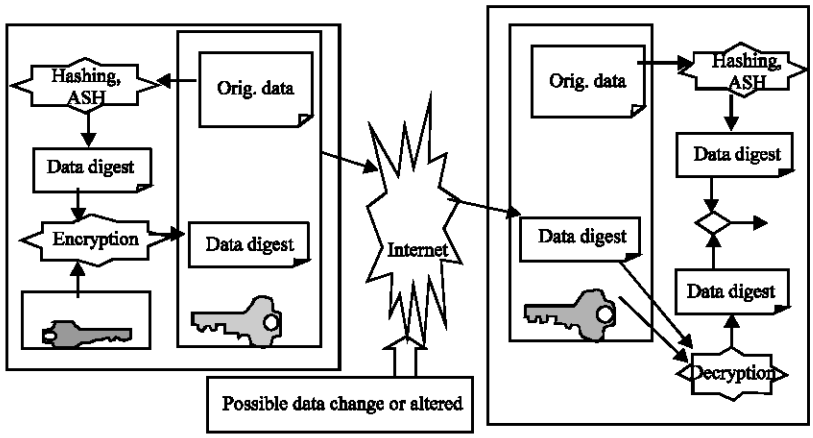


Fig. 6: Non-repudiation concept in web application

- The receiver of the signed data decrypts the message digest with the sender's public key, applies the same hash function to the data and then compares the resulting data digest with the received version.
- Any modification to the data after it was signed will cause the signature verification to fail (integrity).
- If the signature was computed with a private key other than the one corresponding to the public key used for verification, then the verification will fail (authentication).

- Process Patterns _S-process (Signing process)

Forces 1- Reference(s) generation and, 2-Signature generation.

Format

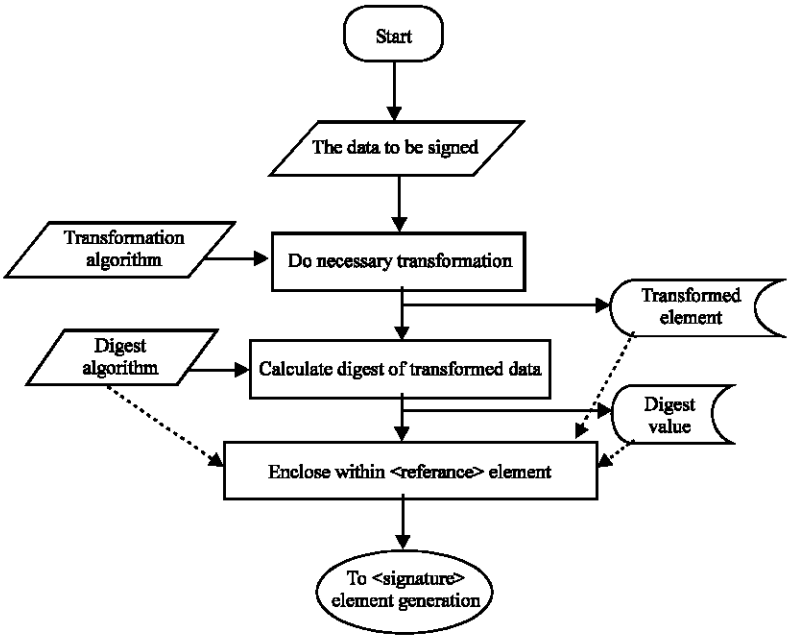


Fig. 7: A single reference element generation process repeated as number of the data objects to be signed; Reference element for each object

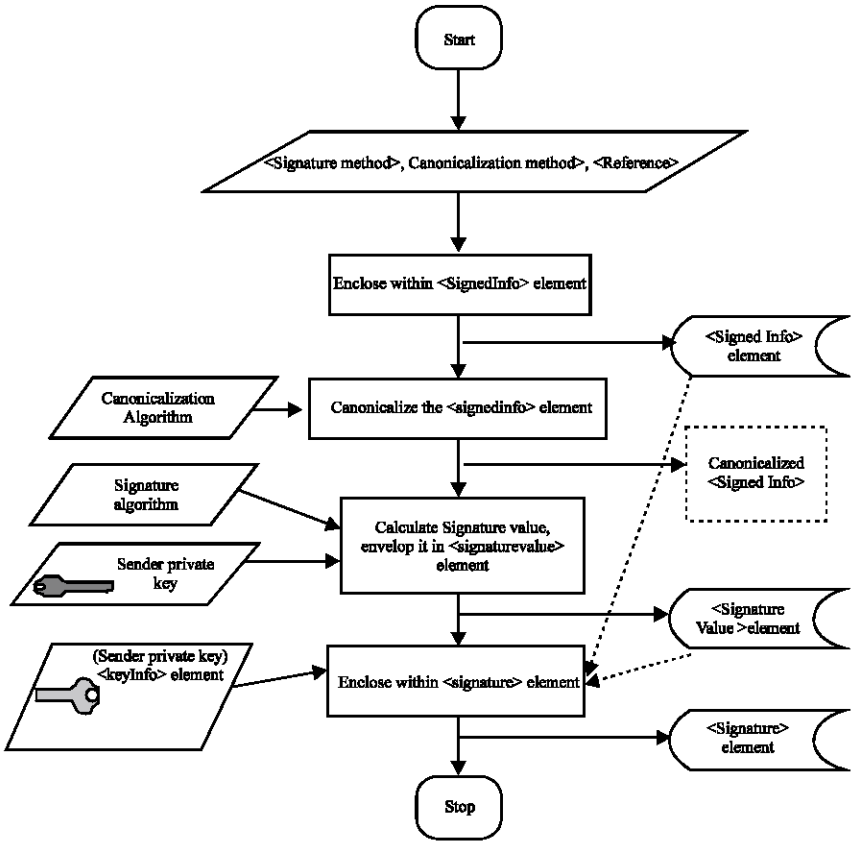


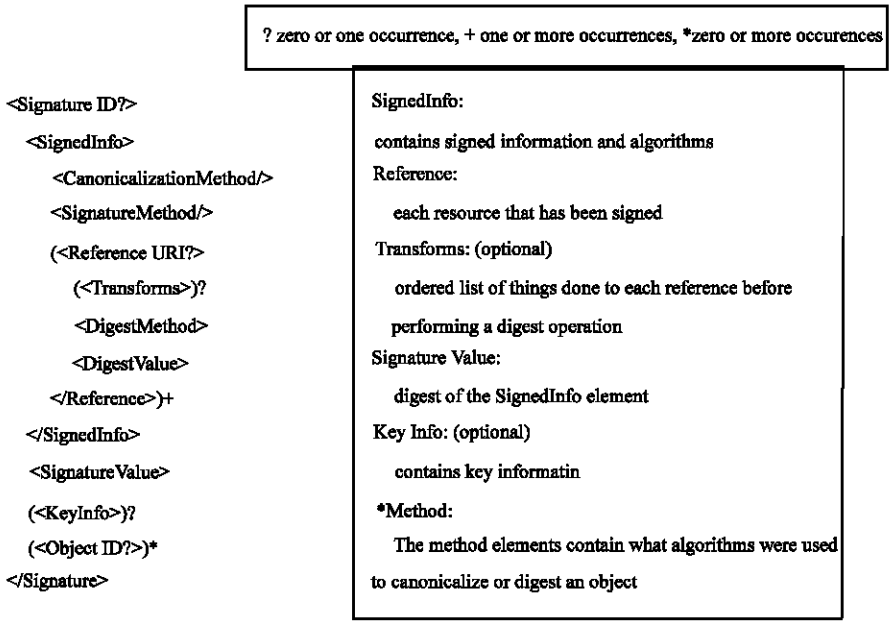
Fig. 8: The signature element generation process; that envelop <signaturevalue> and <KeyInfo> elements. <signature Value> context the digest of canoniclazed <signedInfo> element

XML-Based Security Analysis Pattern (Signing an XML object)

Forces 1 - Generate the <*Signature*> element that is the root element of all standard XML Digital Signatures. It encloses the main elements <SignedInfo> and <SignatureValue> and maybe other optional elements such as <KeyInfo>.

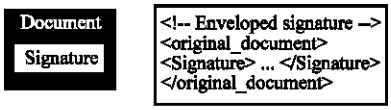
2 - Verify the Signature elements by checking the validation of signature and reference.

Format **Structure**

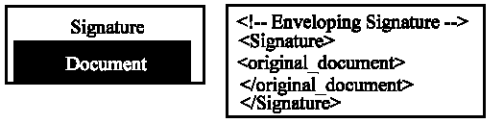


Solution An XML Digital Signature element can reside in one of three different places:

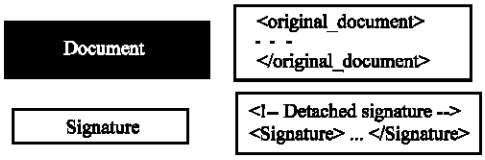
- **Enveloped:** the element is the child element of the signed data (applicable only for singing XML resources).



- **Enveloping:** the element has the signed data (applicable only for singing XML resources)



- **Detached:** the element is completely external to the data being signed; referenced through a URI or a transformation element (suitable for signing resources other than XML)



Creating Signature Elements (at the sender's end) calculate the digest of the <SignedInfo> element and use the public key over it to calculate the value of the <SignatureValue> element. The steps are

• Determine what objects need to be signed (i.e. http://www.anywhere.com/obj), transform them if necessary and then calculate their digests (SHA-1) into DigestValue.

<SignatureMethod Algorithm="URI_of_the_Algorithm" />

```
<Reference URI=" URI_of_the_Element1">
        <! other optional elements i.e. "transforms" elements.....->
  <DigestValue>Knj6lwPOvKtlup4DbeVu8nk=</DigestValue>
 </Reference>
```

- Collect all the references into a single SignedInfo

```
<SignedInfo>
  <Reference URI="URI_of_the_Element1">
        .....
  <DigestValue>Knj6lwPOvKtlup4DbeVu8nk=</DigestValue>
  </Reference>
  <Reference URI=" URI_of_the_Element2">
        .....
  <DigestValue> tlupT4DbeVu8nkKnj6lwPOv=</DigestValue>
  </Reference>
 <SignedInfo>
```

- Canonicalize SignedInfo

```
<CanonicalizationMethod Algorithm=" URI_of_the_Algorithm"/>
```

- Digest SignedInfo using DSA or RSA and place result into SignatureValue

```
<SignatureValue>EPOM~SL*</SignatureValue>
```

- Add additional optional elements as necessary (i.e <keyInfo>)
- Create the <Signature> elements placing at least the main elements <SignedInfo> and <SignatureValue>. It is recommended to add the optional elements like <keyInfo>

```
<signature>
  <signedInfo> Y.</signedInfo>
  <SignatureValue>YY</SignatureValue>
  <keyInfo> Y</keyInfo>
  :
 </signature>
```

Verifying Signature Elements (at the receiver's end) re-calculate and verify the digest of the <SignedInfo> element and its related <SignatureValue> element.

Validation The signature

- Acquire the key from either the KeyInfo element or from an external source
- Canonicalize SignedInfo and then digest it using the acquired key to compare it with the SignatureValue; if they don=t fit, then validation fails

Validation The Reference

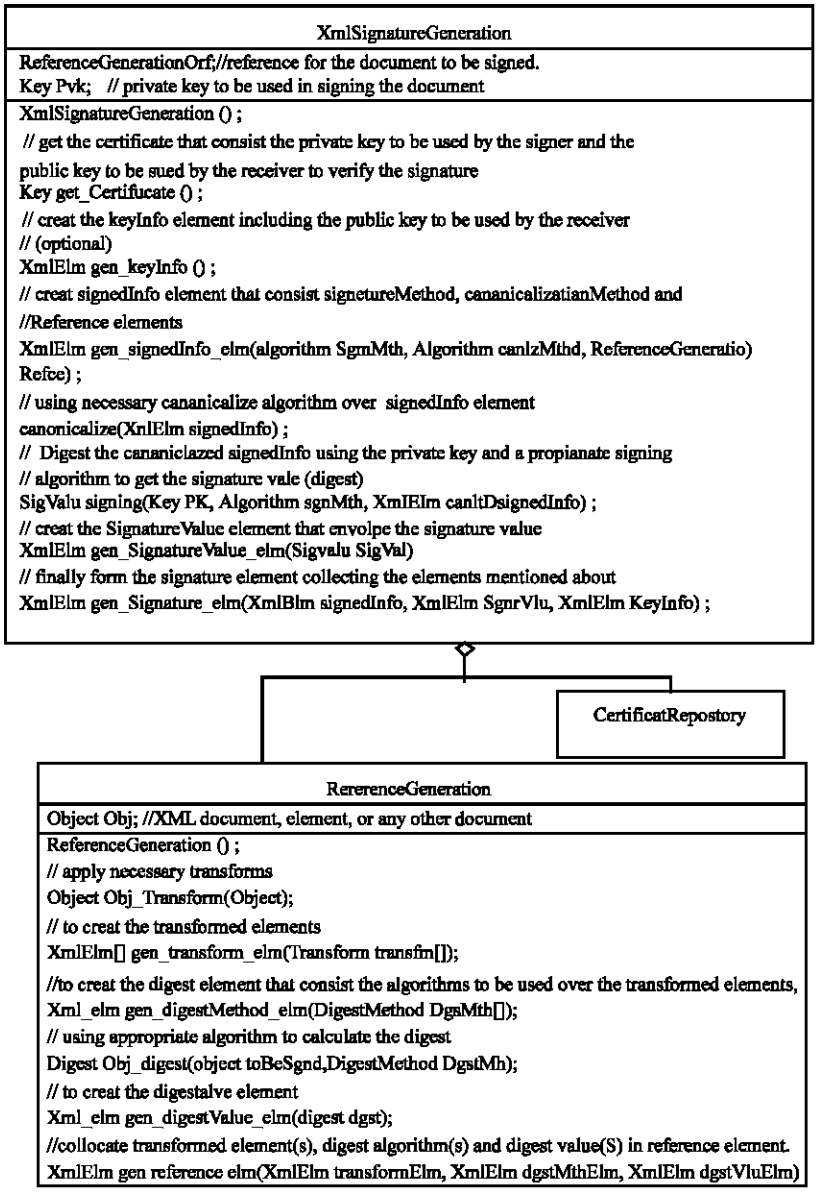
- Canonicalize the SignedInfo element
- For all the Reference elements in SignedInfo
  - Get the object that supposedly was digested
  - Use the Transform elements on objects as needed
  - Compute the digest of the object accordingly
  - Compare the computed value with the DigestValue of that object; if they are different, then validation fails for that object

CONCLUSIONS

This paper first explored the patterns and security patterns concepts and then discussed the description of the patterns language levels and the usage of the language giving sentence example.

XML security patterns are categorized into security of XML document pattern and XML for security function. While the entire are categorized into three abstraction levels, security requirements patterns level, relevant XML-Based security patterns level and the process patterns. The example of sentence presented shows how the patterns from the three levels combination based on the syntax giving a solution for XML security problem from concept to implementation. In the last few years XML became the base technology for many models; that make patterns language to be applicable in several different situations.

Solution



REFERENCES

Alexander, C., 1977. A Pattern Language, Towns, Building, Construction (Center for Environmental Structure Series), Oxford University Press, New York, USA.

Appleton, B., 1997. Patterns and Software: Essential Concepts and Terminology, http://www.enteract.com/~bradapp/docs/patternsintro.html

Gamma, E., R. Helm, R. Johnson and J. Vlissides, 1995. Design patterns: Elements of reusable object-oriented software, Addison-Wesley Longman Inc., Baarn, Holland.

Kreger, H., 2001. IBM, Web Services Conceptual Architecture 1.0, www.ibm-3.com/ software/ solutions/ webservices/pdf/WSCA.pdf.

Lea, D., 1997. Patterns discussion FAQ, http://g.oswego.edu/pd-FAQ.html. Engineering Processes, Proceeding DEXA'99.

Schnmacher, M. and U. Roedig, 2001. Security engineering with patterns. The 8th Conference on Pattern Languages of Programs.

Tawfiq, B., S.S. Zhang, 2005. Groundwork for true XML-based security pattern language. J. Donghua Univ., 8: 12-15.

Vlissides, J., 1998. Pattern Hatching: Design Patterns Applied software pattern series, Addison-Wesley Longman Inc.

Zimmer, W., 1995. Relationships Between Design Patterns. In: Pattern Languages of Program Design. James O. Coplien and Douglas C. Schmidt. (Eds.). Addison-Wesley, pp: 345-364.