P-2584

ANNA UNIVERSITY: CHENNAI 600 025

BONAFIDE CERTIFICATE

Certified that this project report **"SELECTIVE AND AUTHENTIC THIRD PARTY DISTRIBUTION OF XML DOCUMENTS"** is the bonafide work of **"S.GOPINATHAN, Y.NAVEEN BASHA and T.SARAVANAPRABHU"** who carried out the project work under my supervision.

SIGNATURE

Dr. Thangasamy, B.E(Hons)., Ph.D.

DEAN

Dept of Computer Science and Engineering,
Kumaraguru College of Technology,
Coimbatore – 641 006.

SIGNATURE

Mr. E.A.Vimal, M.E.

SUPERVISOR,

Dept of Information Technology,
Kumaraguru College of Technology,
Coimbatore – 641 006.

The candidates with University Register No 71205205019, 71205205033 and 71205205049 were examined by us in the project viva-voce examination held on 28.4.09

INTERNAL EXAMINER

EXTERNAL EXAMINER

DECLARATION

We,

S.GOPINATHAN	**Reg.No:**	**71205205019**
Y.NAVEEN BASHA	**Reg.No:**	**71205205033**
T.SARAVANAPRABHU	**Reg.No:**	**71205205049**

Hereby declare that the project entitled **"SELECTIVE AND AUTHENTIC THIRD PARTY DISTRIBUTION OF XML DOCUMENTS"**, submitted in partial fulfillment to Anna University as the project work of Bachelor of Technology (Information Technology) degree, is a record of original work done by us under the supervision and guidance of Department of Information Technology, Kumaraguru College of Technology, Coimbatore.

Place: Coimbatore

Date:

S.Gopinathan Y.Naveen Basha T.Saravanaprabhu

[S.Gopinathan] [Y.Naveen Basha] [T.Saravanaprabhu]

Project Guided by

[Mr. E.A.Vimal M.E]

ACKNOWLEDGEMENT

We express our sincere thanks to our Chairman **Padmabhushan Arutselvar Dr. N. Mahalingam B.Sc., F.I.E.,** Vice Chairman Dr.K. **Arumugam B.E., M.S., M.I.E.,** Correspondent **Shri.M.Balasubramaniam** and Joint Correspondent **Dr.A.Selvakumar** for all their support and ray of strengthening hope extended. We are immensely grateful to our Vice Principal **Prof R.Annamalai,** for his invaluable support to the outcome of this project.

We are deeply obliged to **Dr.S.Thangasamy,** Dean, Department of Computer Science and Engineering for his valuable guidance and useful suggestions during the course of this project.

We also extend our heartfelt thanks to our project co-ordinator, **Ms. L.S Jayashree M.E., (Ph.D.),** Associate Professor, Department of Information Technology for providing us her support which really helped us.

We are indebted to our project guide and extend our gratitude towards **Mr. E.A.Vimal M.E.,** Lecturer, and Department of Information Technology for his helpful guidance and valuable support given to us throughout this project.

We thank the teaching and non-teaching staffs of our Department for providing us the technical support during the course of this project. We also thank all of our friends who helped us to complete this project successfully.

ABSTRACT

Third-party architectures for data publishing over the Internet today are receiving growing attention, due to their scalability properties and to the ability of efficiently managing large number of subjects and great amount of data. In third-party architecture, there is a distinction between the Owner and the Publisher of information. The Owner is the producer of information, whereas Publishers are responsible for managing (a portion of) the Owner information and for answering subject queries. A relevant issue in this architecture is how the Owner can ensure a secure and selective publishing of its data, even if the data are managed by a third-party, which can prune some of the nodes of the original document on the basis of subject queries and access control policies. An approach can be that of requiring the Publisher to be trusted with regard to the considered security properties. However, the serious drawback of this solution is that large Web-based systems cannot be easily verified to be secure and can be easily penetrated.

For these reasons, we propose an alternative approach, based on the use of digital signature techniques, which does not require the Publisher to be trusted. The security properties we consider are authenticity and completeness of a query response, where completeness is intended with regard to the access control policies stated by the information Owner. In particular, we show that, by embedding in the query response one digital signature generated by the Owner and some hash values, a subject is able to locally verify the authenticity of a query response. Moreover, we present an approach that, for a wide range of queries, allows a subject to verify the completeness of query results.

CONTENTS

TABLE OF CONTENTS

LIST OF FIGURES:

LIST OF ABBREVIATION

XML	Extentisible Markup Language
IIS	Internet Information Server
XSLT	XML Style Sheet
DTD	Document Type Definition
ASP	Active Server Page
SOAP	Simple Object access Protocol
W3C	Wide Web Consortium

INTRODUCTION

1. INTRODUCTION

1.1 GENERAL:

XML (eXtensible Markup Language) is rapidly becoming a de facto standard for document representation and exchange over the Web. A common requirement for Web applications is thus the need for secure publishing of XML documents. By secure publishing, we mean that the publishing service must ensure some security properties to the data it manages. Third-party architectures for data publishing over the Web are today receiving growing attention, due to their scalability properties and to the ability of efficiently managing a large number of subjects and a great amount of data. In third-party architecture, there is a distinction between the Owner and the Publisher of information. The Owner is the producer of the information, whereas Publishers are responsible for managing (a portion of) the Owner information and for answering subject queries. A relevant issue in this architecture is how the Owner can ensure a secure publishing of its data, even if the data are managed by a third-party.

1.1.1 EXISTING SYSTEM:

In existing system the integrity check is performed for the entire XML document. When the user queries part of the document, integrity cannot be checked for partial documents. If we need to check the integrity then the entire document has to be provided. If the information is sensitive then these methods fails to check the integrity. The document which contains partial data cannot be checked for integrity. The most intuitive solution is that of requiring publishers to be trusted with regard to the considered security properties.

Drawbacks:

- Publishers may publicly release or sell the confidential information of the firm.
- We cannot ensure that data received by the subjects are un-modified and original
- Large Web-based systems cannot be easily verified to be secure and can be easily penetrated.

1.1.2 PROPOSED SYSTEM:

In proposed system we using an algorithm called Merkle Hash Algorithm and RSA which provides both authentication and integrity of the XML document. The subject can himself verify these properties.

- RSA
- Access Control Policies
- Policy Configuration
- Digital Signature and Hashing

Advantages:

- Publishers need to be trusted with respect to authenticity and completeness.
- Subjects can access the documents , according to access control policies only
- Our system can be applied to any decentralized architecture and our solution offers the advantage of being scalable and of reducing the risk that the Owner becomes the bottleneck of the entire system

1.3 OBJECTIVE OF THE PROJECT:

The key point of our approach is that, even though we do not require the Publisher to be trusted with respect to authenticity and completeness properties, we are able to ensure, that a subject is able to verify such properties on the answer returned by a Publisher. This capability is obtained by a combination of digital signature and hashing techniques. Let us first consider authenticity. In the scenario, we consider that it is not enough that the Owner signs each document it sends to the Publisher since the Publisher may return to subject only selected portions of a document, depending on the query the subject submits and on the access control policies in place. For this reason, we propose an alternative solution that requires that the Owner sends the Publisher, in addition to the documents it is entitled to manage, a summary signature for each managed document, generated using a technique based on Merkle hash trees. The idea is that, when a subject submits a query to a Publisher, the Publisher also sends him the signatures of the document, besides the query result.

LITERATURE REVIEW

2. LITERATURE REVIEW

2.1 Feasibility Study:

Ensuring document authenticity means that the subject receiving a document is assured that its contents are actually original from the Owner itself. Ensuring the completeness of the response means that any subject must be able to verify that he or she has received all or portions of the document that is he/she requested to access, according to the stated access control policies.

2.1.1 Owner-Publisher Interaction:

Owner sends the XML document and secure structure, in addition to the documents it is entitled to manage, a summary signature for each managed document, generated using a technique based on Merkle hash trees. For each document the Publisher is entitled to manage, the Owner sends the corresponding security enhanced (SE-XML) document and the corresponding secure structure, along with the access control policies and policy configuration for each element (node) of XML tree. Interaction is shown in the fig.2.1.1.

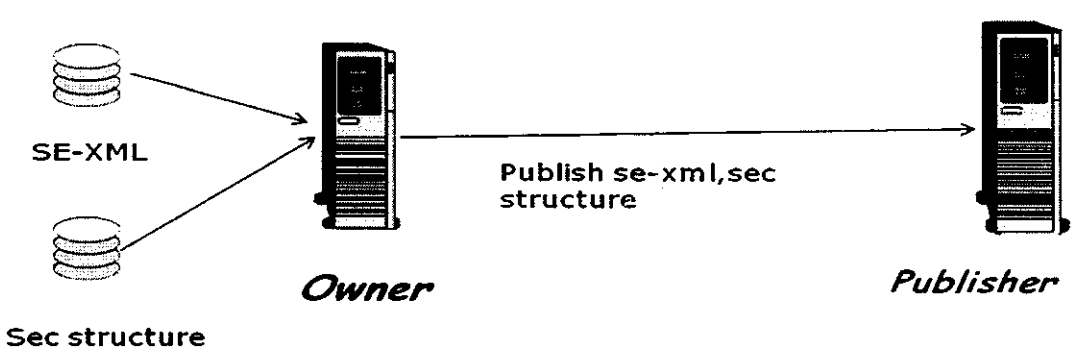


Fig 2.1.1 Owner – Publisher Interaction

2.1.2 Subject- Publisher Interaction:

When a subject submits a query to a Publisher, the Publisher sends him the signature of the document besides the query result. The Interaction between subject and publisher is shown in the fig. 2.1.2. In this way, the subject can locally recomputed the same bottom- up hash value signed by the Owner and, by comparing the two values, he/she can verify whether the Publisher has altered the content of the query answer and can be sure of its authenticity.

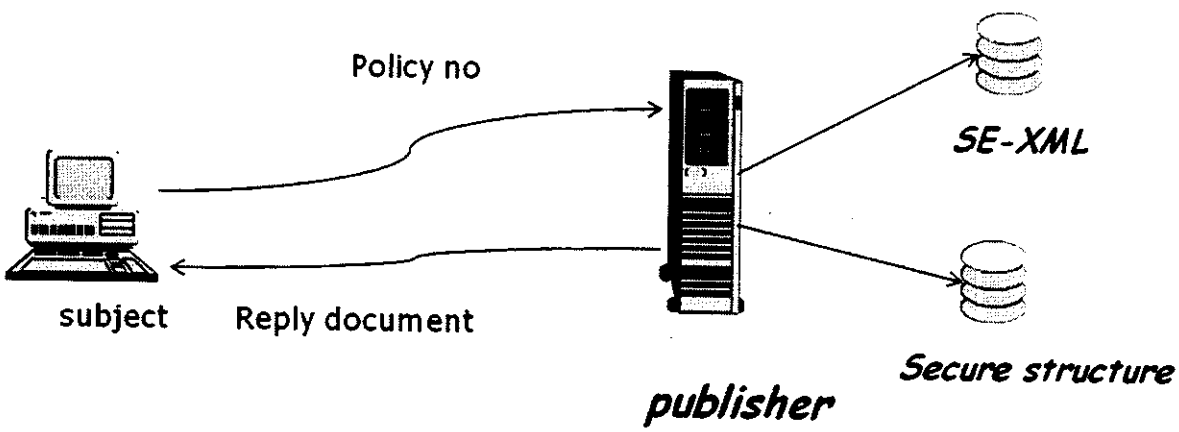


Fig 2.1.2 Subject – Publisher Interaction

2.1.3 Access Control Policy:

Each subject has some access control policy like some documents or portions of some documents are accessible only to some of the subjects. This is called Selective property. Query answers return the Publisher to a subject are filtered according to the access control policies specified by the owner. Thus the receiving subject is prevented from accessing unauthorized information, being at the same time able to perform the completeness verification.

Policy Configuration:

When a subject subscribes to the Owner, it receives back a set of policies called subject policy configuration, providing information on the access control policies that the subject satisfies.

Reply Generation:

When a subject submits a query to a publisher, the publisher first determines the set of nodes that need to be returned to him/her. Such nodes are determined by evaluating the query on the SE-XML version of the requested document and by pruning from the set of nodes returned by the evaluation, those nodes corresponding to portions for which subject does not possess appropriate authorizations.

Subject Verification:

A subject upon receiving a reply document and a secure structure can verify the authenticity and the completeness of the corresponding query answer.

2.1.4 Owner- Subject Interaction:

To make the publisher able to verify which access control policies apply to a subject, the owner returns the subject a policy configuration i.e., a certificate containing information about the access control policies that apply to the subject. The Interaction between Owner and subject is shown in the fig 2.1.3 .The subject policy configuration is signed with the private key of the owner to prevent the subject from altering its content. Thus, all the information exchanged between the parties of our architecture is completely secure.

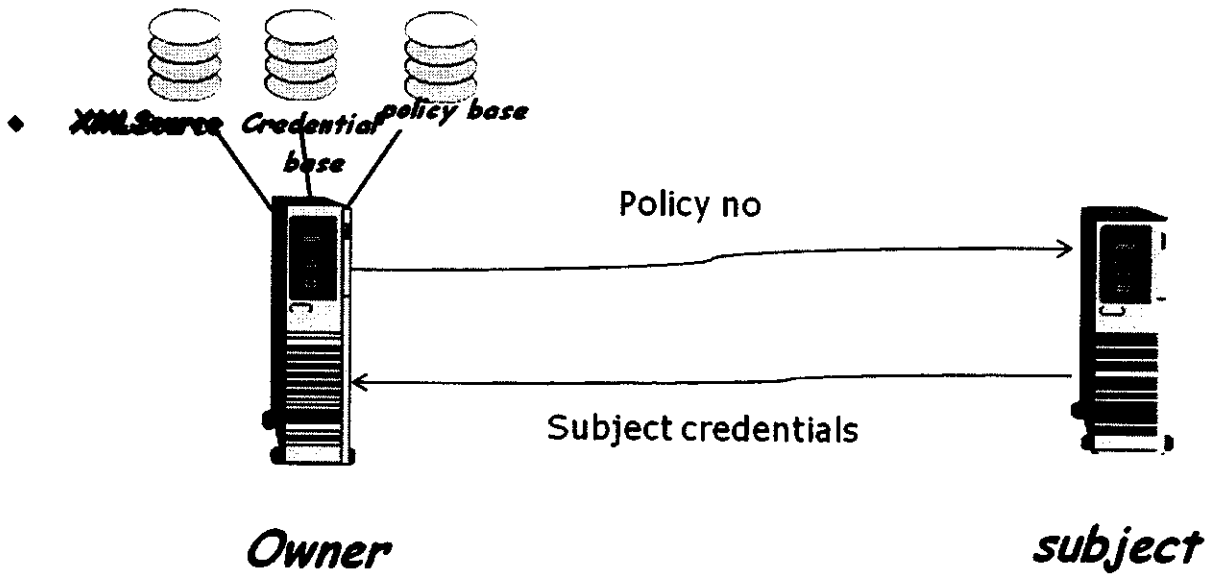


Fig 2.1.3 Owner –Subject Interaction

2.2 HARDWARE REQUIREMENTS (MINIMUM)

HARD DISK	:	40 GB
MAIN MEMORY	:	512 MB RAM
PROCESSOR	:	PENTIUM IV 2.4 GHz
KEYBOARD	:	105 KEYS

2.3 SOFTWARE REQUIREMENTS

OPERATING SYSTEM	:	WINDOWS XP
LANGUAGE	:	VB.NET 2008
WEB SERVER	:	IIS
BROWSER	:	COMPATIBLE BROWSERS
XML EDITOR	:	XML MARKER 1.1

2.4 SOFTWARE OVERVIEW:

2.4.1 VB.NET

Working With Forms and Controls:

Forms allow us to work visually with controls and other items from the tool box .In VB.NET forms are based on the System.Windows.Forms namespace and the form class based on System.Forms.Form. A Control is an object that can be drawn on to the form to enable or enhance user interaction with the application.

Working with Menus:

In this project, we used to control for three main navigations namely, Owner, Publisher, and Subject. Owner menu has sub navigation through which we can access the forms of Policy Master, Subject Configuration as sub navigation in which we can enter the details and store in the XML file. Likewise all other navigations are tied up with their corresponding forms through sub menus.

2.4.2 .NET FRAMEWORK

- The .NET Framework is the infrastructure for the new Microsoft .NET Platform.
- The .NET Framework is a common environment for building, deploying, and running Web applications and Web services.
- The .NET Framework contains a common language runtime and common class libraries like ADO.NET, ASP.NET and Windows Forms to provide advanced standard services that can be integrated into a variety of computer systems.

- The .NET Framework provides a feature rich application environment, simplified development and easy integration between a numbers of different development languages.

Common language runtime:

One of the design goals of .NET Framework was to unify the runtime engines so that all developers could work with a set of runtime services. The .NET Framework is based on the Common Language Runtime (CLR). The CLR provides capabilities such as memory management, security, and robust error handling to any language that work with the .NET Framework.

2.4.3 .NET CLASS LIBRARIES

The .NET Framework provides many classes that help developers re-use code. The .Net class libraries contain code for programming topics such as threading, file I/O database support, XML support, XML parsing .Data structures such as stacks and queues are also supported. This entire class library is available to any programming language that supports the .NET Framework.

DETAILS OF METHODOLOGY EMPLOYED

3. DETAILS OF METHODOLOGY INVOLVED

Introduction:

Building blocks of XML documents are nested, tagged elements. Each tagged element has zero or more sub elements; zero or more attributes, and may contain textual information (data content). Elements can be nested at any depth in the document structure. Attributes can be of different types, allowing one to specify element identifiers (attributes of type ID), additional information about the element (e.g., attributes of type CDATA containing textual information), or links to other elements of the document (attributes of type IDREF(s)/URI(s)).

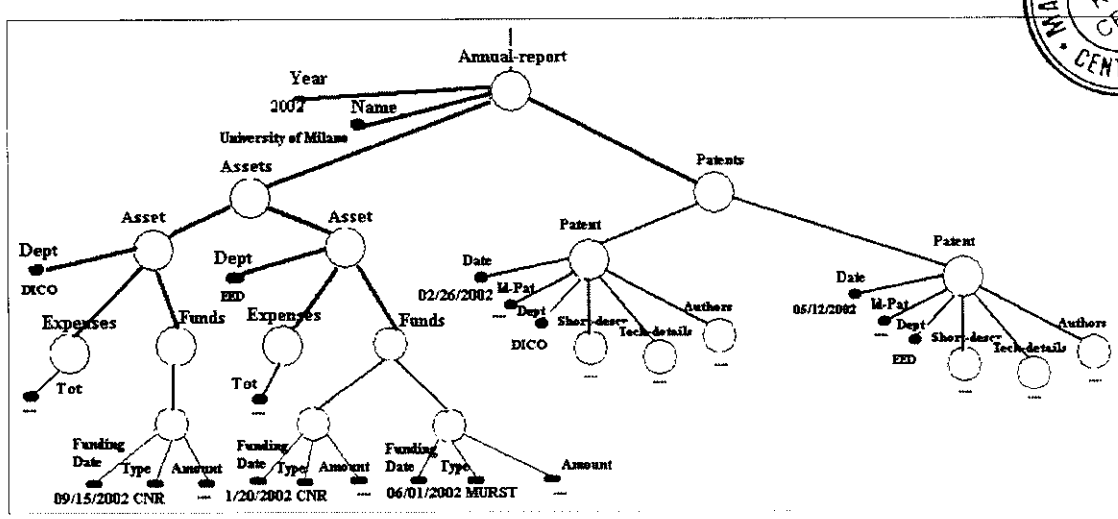


Fig 3.1.1 XML Design

Based on the above definition, an XML document can be represented as a graph. In the graph representation adopted, white nodes represent elements, whereas black nodes represent attributes. The graph representation contains edges representing the element-attribute and the element-sub element relationships, and link edges, representing links between elements as in fig 3.1.1. Solid lines represent edges, whereas dashed lines represent link edges.

3.2. Merkle Hash Trees for XML Document:

Authenticity is ensured by using the Merkle tree authentication mechanism proposed in and adapting it to XML. The method we propose allows a subject to prove source authenticity as well as the authenticity of both the schema and the content of a document. To accomplish this goal, the idea is to associate a hash value with each node in the graph representation of an XML document. More precisely, the hash value associated with an attribute is obtained by applying a hash function over the concatenation of the attribute value and the attribute name. By contrast, the hash value associated with an element is the result of the same hash function computed over the concatenation of the element content, the element tag name, and the hash values associated with its children nodes, both attributes and elements. Hash values associated with the nodes of an XML document are computed by the Merkle hash function.

3.3. Access Control Model for XML documents:

Access control policies are specified by the Owner according to the access control model, whose main characteristics are summarized in what follows. In this model, subjects are qualified by means of credential. A credential is a set of properties concerning a subject that are relevant for security purposes (for example, age, and position within an organization). Credentials are encoded using an XML-based language, called X-Sec. Access control policies specify conditions on the credentials and properties of the credentials, using an XPath-compliant language. The access control model provides varying access granularity levels, and can express policies that apply to: 1) all the instances of a DTD/XML Schema, 2) collections of documents not necessarily instances of the same DTD/XML Schema, and 3) selected portions within a document(s), or a link (or a set of links). This set of granularity levels is complemented with the possibility of specifying access control policies based on the document content, in addition to the document structure. Like credentials, access control policies are also encoded using the X-Sec language. We use the term Policy base to denote the XML file encoding the access control policies of the Users using XPath.

Credential Base:

A credential base is one which stores user information such as their position within the organization along with other information that is used to distinguish each subject from other.

Credential XML:

<cred_id>

<username\>

<password\>

<department\>

<company\>

<cred_id>

Policy Base:

A policy base stores information which is used to point to specific XML nodes of the document. We associate one or more credential to these policies which means that the user can access the nodes accessed by the policies. The reason for separating the policies and credential is that whenever owner changes users the owner can directly change in the credential base which will not be reflected in the policy base. Thus it minimizes unwanted updating of policies.

XPath:

XPath is the result of an effort to provide a common syntax and semantics for functionality shared between XSL Transformations and XPointer. The primary purpose of XPath is to address parts of an XML document. In support of this primary purpose, it also provides basic facilities for manipulation of strings, numbers and Booleans. XPath uses a compact, non-XML syntax to facilitate use of XPath within URIs and XML attribute values. XPath operates on the abstract, logical structure of an XML document, rather than its surface syntax. XPath gets its name from its use of a path notation as in URLs for navigating through the hierarchical structure

of an XML document. In addition to its use for addressing, XPath is also designed so that it has a natural subset that can be used for matching (testing whether or not a node matches a pattern); this use of XPath is described in XSLT .XPath models an XML document as a tree of nodes. There are different types of nodes, including element nodes, attribute nodes and text nodes.

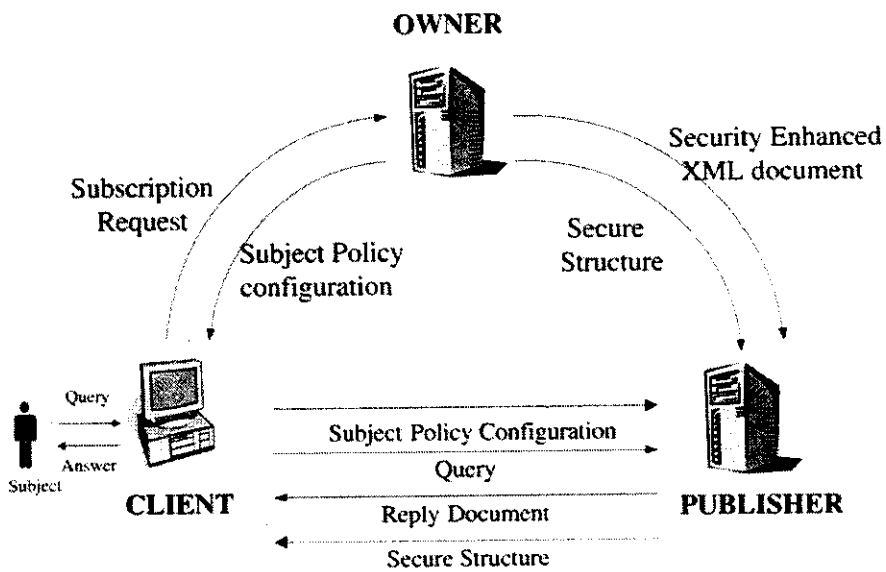


Fig 3.2.1 System Architecture

SUBJECT – OWNER INTERACTION:

When a subject subscribes to the Owner, it receives back a set of policies called subject policy configuration, providing information on the access control policies that the subject satisfies by a web service. The overall architecture is presented in fig 3.2.1

Web service:

A Web service (also Web Service) is defined by the W3C as "a software system designed to support interoperable machine-to-machine

interaction over a network". Web services are frequently just WebAPIs that can be accessed over a network, such as the Internet, and executed on a remote system hosting the requested services. It uses SOAP protocol for communication.

OWNER – PUBLISHER INTRACTION:

For each document the Publisher is entitled to manage, the Owner sends the corresponding security enhanced document and the corresponding secure structure, which are described in the following sections.

Security Enhanced XML Document:

The first information the security enhanced document contains is which access control policies apply to the corresponding document. Policy information is specified at the element level since different access control policies can apply to different elements and/or attributes of the same document. The idea is to encode information about the set of policies that apply to a specific element into a string of hexadecimal values, called policy configuration, and to store this string as an additional attribute of the corresponding element within the security enhanced version of the document (this attribute is called Policy).

Secure Structure:

To prove completeness of XML queries, a subject receives from the Publisher the secure structure of the XML documents on which the query is performed. In this section, we show how the secure structure is generated. The basic idea is to supply the subject with the structure of the XML document on which the query is submitted where, with the term structure,

we mean the XML document without data contents, that is, containing only the names of the tags and attributes of the XML document. The subject is then able to locally perform on the structure all queries whose conditions are against the document structure of the original document. Thus, the subject can match the result with the answer sent by the Publisher. Obviously, in such a basic approach, the subject is able to view the tag and attribute names of the whole document and thus also those referring to portions he/she may not be authorized to access. To overcome this drawback, the secure structure of the XML document is generated by hashing with a standard hash function each tag and attribute name. Since the value returned by the hash function may contain characters not allowed in an XML well-formed document. During the generation of the secure structure the resulting hash values are encoded into an integer-based representation. Moreover, to be compliant with the XML syntax, we insert symbol "x" as a prefix of the integer before storing it as a tag name. Additionally, to extend the set of queries for which it is possible to prove completeness, we also insert the hashed attribute values of the XML document in the secure structure. The node-set returned by evaluating a query on the secure structure could be a superset of the nodes the subject is entitled to see according to the Owner access control policies. Thus, in order to verify the completeness, the subject must also consider the access control policies specified on the document. For this reason, the secure structure contains also the Policy, whose tag name and content are not hashed. Additionally, in order to prevent alterations by the Publisher, the Owner computes the Merkle Signature of the secure structure.

SUBJECT – PUBLISHER INTERACTION

When a subject s submits a query to a Publisher, the Publisher first determines the set of nodes that need to be returned to s. Such nodes are determined by evaluating the query on the SE-XML version of the requested document(s) and by pruning from the set of nodes returned by the evaluation, those nodes corresponding to portions for which s does not possess appropriate authorizations. Information on access control policies that apply to s are transmitted by s to the Publisher when submitting the access request. More precisely, the subject sends the Publisher his/her policy configuration together with the submitted query. Then, the set of nodes to be returned to is complemented with additional information that is used by s to authenticate the answer and to verify its completeness. In particular, all the additional information needed to verify the authenticity, as well as the nodes of the requested document(s) to be returned to the subject, are organized into an XML document, called reply document.

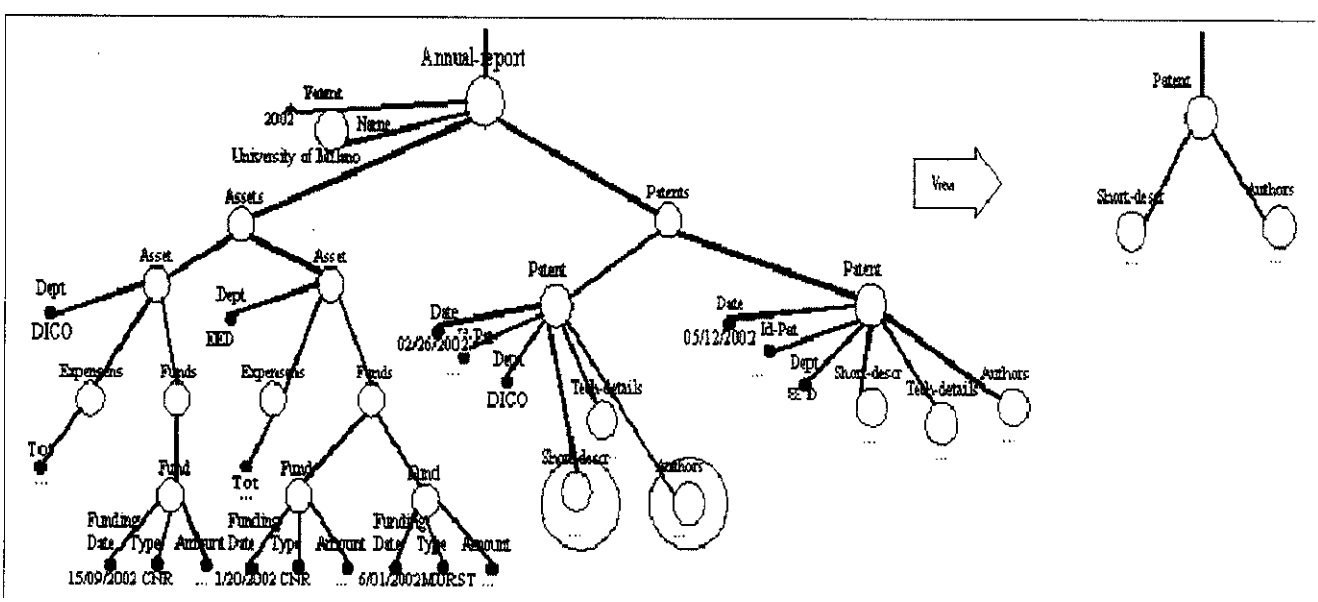


Fig 3.2.2 Selective Query

3.3. Reply Document Generation:

For the sake of simplicity, in the following, we consider only queries on a single document. The methods we present, however, can be easily extended to queries referring to multiple documents. In general, an answer to a query may contain only selected portions of a given XML document. For instance, consider the XML document in Fig.3.1.1, and suppose that an EED professor submits a query q asking for all DICO patents. According to the access control policies in fig 3.2.2, the nodes answering q and for which an EED professor has the necessary authorizations are the Short-description and Authors elements. We create a function, called view, which takes as input a query q and the policy configuration of a subject s, and turns the set of nodes answer to q and for which s has the necessary authorizations into a well-formed XML document

Algorithm 1: The Reply Generator Algorithm

INPUT: 1. The SE-XML version $g = (V_g, \bar{v}_g, E_g, \phi_{E_g})$ of an XML document d; 2. A query q on d submitted by s; 3. $PC(s)$ the policy configuration of s

OUTPUT: The reply document r

1. $PolConf$ is initialized to be empty
2. Let $r = (V_r, \bar{v}_r, E_r, \phi_{E_r})$ be the XML document returned by function $Evaluate(q)$
3. **For** each $v \in V_r$:
   - A Let $v^g \in V_g$ be the node corresponding to v in document g
   - B **If** $v^g \in V_g^c$: **For** each $c \in v$.PC: $PolConf := PolConf || HexToBin(c)$
   - **else**:
     - a Let w^g be the father of v^g
     - b Let j be the relative order of v^g wrt its siblings
     - c Let $PolString$ be the j-th policy string extracted from $w^g.PC_{Attr}$
     - d **For** each $c \in PolString$: $PolConf := PolConf || HexToBin(c)$
   - **endif**
   - C $Find := 0$
   - D **For** $n = 1$ to $length(PolConf)$:
     - a $Bit := nextBit(PolConf)$
     - b **If** $Bit := 1$:
       - i Let y be the n-th identifier stored in `Policy` element
       - ii **If** y appears in $PC(s)$: $Find := 1$, Break
   - **endfor**
   - E **If** $Find = 0$: Remove v from V_r
   - F $PolConf$ is initialized to be empty

   endfor
4. **For** each $a \in V_r^a$:
   - A Create a new node e with tag `AttributeElement`
   - B $e.content := a.val$
   - C Create a new attribute in e with name `AttrName`
   - D e.`AttrName`$.val := a.name$
   - E Add e to V_r, remove a from V_r

   endfor
5. **For** each $w \in V_r$:
   - A Let $w^g \in V_g$ be the node corresponding to w in document g
   - B **If** $w = \bar{v}_r$: $v^g = \bar{v}_g$
   - **else**:
     - a Let v be the father of w in document r
     - b Let $v^g \in V_g$ be the node corresponding to v in document g
   - **endif**
   - C Add attribute `MhPath` to node w
   - D w.`MhPath` $:= MhPath(w^g, v^g)$

   endfor
6. Add the attribute `Sign` to $\bar{v}_r$, and set its value equal to attribute `Sign` in document g
7. $\bar{r}$:=ReBuild(r)
8. **Return**($\bar{r}$)

3.4 Subject hash verification:

In this section, we illustrate how a subject, upon receiving a reply document and a secure structure, can verify the authenticity and the completeness of the corresponding query answer. This is an algorithm for verifying the authenticity of a query answer. The algorithm takes as input a reply document r, returned as answer to a query on document d, and returns true if all the elements in the reply document are authentic. It returns False, otherwise. Starting from each terminal node in the reply document, the algorithm re computes the Merkle hash value of the root of d through a bottom-up computation that uses the values of attributes MhPath of each node belonging to the path connecting the terminal node to the root of the reply document. In the algorithm, there is the need of knowing where the considered element is located with regard to its siblings, in order to correctly compute its Merkle hash value. Due to this reason, in the Merkle hash path, we store this information also. To do that, we use symbol "own" to denote a subsequent level in the hierarchy, and symbol "hash" to denote the position of the element with regard to its siblings. Then, the obtained value is compared with the decryption of the Merkle Signature of d, using the Owner public key (the Merkle Signature of d is stored into attribute Sign): If the two values coincide, then all the nodes belonging to the path are authentic. Otherwise, the algorithm terminates and returns False.

INPUT: A reply document $r = (V_r, \bar{v}_r, E_r, \phi_{E_r})$

OUTPUT: $True$ if all the nodes in r are authentic, $False$ otherwise

1 Let $V_T = \{v | v \in V_r^s$ AND each children of v belongs to $V_r^a\}$

2 **For** each $v_t \in V_T$:

A **If** $v_t.tagname =$ `AttributeElement`: $MyMerkle := h(h(v_t.content)||h(v_t.\texttt{AttName}.val))$

else: $MyMerkle := h(v_t.content)||h(v_t.tagname)$

B **For** each $w \in Path(v_t)$:

a **While** ((*curr-level*:=*Extract-next-level*(w.`MhPath`))$\neq 0$):

i) **While** ((*MhX*:=*Extract-next-Merkle*(curr-level))$\neq 0$):

a **If** $MhX =$'#': $Mhx := MyMerkle$

b $Merkle := Merkle||MhX$

endwhile

ii) $Merkle := h(Merkle)$

endfor

C **If** $(Merkle \neq D_{KU_{Owner}}[\texttt{Sign}.val])$: **Return**($False$)

endfor

3 **Return**($True$)

3.5. Subject Completeness Verification:

In general, queries on an XML document can be classified into two groups: queries depending on the structure of the XML document, and queries depending on the content. The proposed solution for completeness verification implies the translation of the query q, by substituting the tag/attribute names and attribute values with the corresponding hash values and policies. Afterward, the translated query can be evaluated on the secure structure. In such a way, under the assumption of a collision-resistant hash function, the node-set resulting by the evaluation of the query on the secure structure corresponds to all and only the nodes of the requested document, answering query q. the last step of completeness verification is to hash the answer received by the Publisher, and match it with the obtained node-set.

PERFORMANCE EVALUATION

4. PERFORMANCE EVALUATION:

Testing:

Testing is a process of checking whether the developed system is working according to the original objectives and requirements.Testing is a set of activities that can be planned in advance and conducted systematically. Testing is vital to the success of the system. System testing makes a logical assumption that if all the parts of the system are correct, the global will be successfully achieved. In adequate testing if not testing leads to errors that may not appear even many months. This creates two problems,

- The time lag between the cause and the appearance of the problem.
- The effect of the system errors on the files and records within the system.

A small system error can conceivably explode into a much larger Problem. Effective testing early in the purpose translates directly into long term cost savings from a reduced number of errors. Another reason for system testing is its utility, as a user-oriented vehicle before implementation. The best programs are worthless if it produces the correct outputs. No other test can be more crucial. Following this step, a variety of tests are conducted.

- Unit testing
- Integration testing
- Validation testing

4.1 Unit Testing:

A program represents the logical elements of a system. For a program to run satisfactorily, it must compile and test data correctly and tie in properly with other programs. Achieving an error free program is the responsibility of the programmer. Program testing checks for two types of errors: syntax and logical. Syntax error is a program statement that violates one or more rules of the language in which it is written. Each and every module i.e. all the four modules have been tested and found to be free of logical and syntax errors.

4.2 Integration Testing

All the four modules have been integrated one by one and testing have been performed at various levels. Logical errors appeared at various levels and they have been cleared. The project is tested to come out with the expected output.

4.3 Validation Testing:

All the inputs entered to each of the module are validated to limit unnecessary inputs to the project such as giving a string in place of integer. Validations to all the input boxes are tested.

CONCLUSION

5. CONCLUSION

With a set of digital signatures generated by the Owner and no trust required for the Publisher, we have shown that a subject can verify the authenticity of a query response. Additionally, for a wide range of XPath queries, a subject is also able to verify the completeness of a query result, with respect to the access control policies stated by the information Owner. This is obtained using secure structures. Making a distinction between the Owner and the Publisher offers two benefits. First, in any decentralized architecture, such a solution offers the advantage of being scalable and of reducing the risk that the Owner becomes the bottleneck of the entire system. Second, this architecture does not require the Publisher to be trusted, with respect to document authenticity and completeness.

FUTURE ENHANCEMENTS

6. FUTURE ENHANCEMENT

This project ensures authenticity, integrity and completeness of the XML document. In the future confidentiality (i.e., encrypting the XML data) can also be included so that the publisher cannot view it. Since it depends on a single secret key, the confidentiality is compromised when publisher gets it from subject. Therefore if possible many secret keys for different subjects will ensure confidentiality. Managing updates in a running application is one of the difficult problems that web application manager's face. With a single package like SQL which manages updates from different users, but for an xml file synchronization between various threads has to be maintained. Further updates are managed along the sub tree and root hash has to be digitally signed once again. To do all this requires some sort of unique programming technique.

Since, in the proposed framework, a modification on a document also implies an update of the security – enhanced version and of the secure structure of the document, a key issue is the efficient management of updates. To this purpose for each kind of update that could occur over the owner s XML source or the Policy Base , the corresponding updates that the owner has to perform on the security enhanced version and on the secure structure of the involved documents. There are two main kinds of updates that need to be considered. The first is an update of the PB such as for instance, an access control policy insertion, deletion, or update. In the case of policy insertion, it is necessary to update the SE-XML version and the secure structure of all the documents to which the new policy applies.

APPENDIX

6. APPENDIX

6.1 SAMPLE SOURCE CODE:

6.1.1 Creating SE-XML:

Hash Generation.aspx.vb:

```
Partial Class merklehash
    Inherits System.Web.UI.Page
    Dim xdwr As System.Xml.XmlDocument = New System.Xml.XmlDocument()
    Public Function hashcode(ByVal list As System.Xml.XmlNodeList, ByVal parent As
System.Xml.XmlElement) As String
        Dim str, attstr, ret, owe As String
        Dim att As System.Xml.XmlAttributeCollection
        Dim attr As System.Xml.XmlAttribute
        Dim node As System.Xml.XmlNode
        Dim add As System.Xml.XmlElement
        ret = ""
        Dim text As System.Xml.XmlText
        For Each node In list
            If node.NodeType = System.Xml.XmlNodeType.Element Then
                If node.HasChildNodes = True And node.ChildNodes.Count > 1 Then
                    add = xdwr.CreateElement(node.Name)
                    text = xdwr.CreateTextNode(node.Value)
                    add.AppendChild(text)
                    owe = hashcode(node.ChildNodes, add)
                    att = node.Attributes
                    attstr = ""
                    For Each attr In att
                        If attr.Name <> "policy" Then
                            attstr = attstr & getSHA1Hash(getSHA1Hash(attr.Name) &
getSHA1Hash(attr.Value))
                        End If
```

```
            add.SetAttribute(attr.Name, attr.Value)
          Next
          If node.Name <> Nothing Then
            str = getSHA1Hash(node.Name)
          End If
          str = getSHA1Hash(str & attstr)
          add.SetAttribute("own", str)
          str = getSHA1Hash(str & owe)
          add.SetAttribute("hash", str)
        Else
          add = xdwr.CreateElement(node.Name)
          att = node.Attributes
          attstr = ""
          For Each attr In att
            If attr.Name <> "policy" Then
              attstr = attstr & getSHA1Hash(getSHA1Hash(attr.Name) &
getSHA1Hash(attr.Value))

            End If
            add.SetAttribute(attr.Name, attr.Value)
          Next
          If node.Name <> Nothing Then
            str = getSHA1Hash(node.Name)
          End If
          If node.InnerText <> Nothing Then
            str = getSHA1Hash(str & getSHA1Hash(node.InnerText))
          End If
          str = getSHA1Hash(str & attstr)
          add.SetAttribute("hash", str)
          text = xdwr.CreateTextNode(node.InnerText)
          add.AppendChild(text)
```

```
            End If
            ret = ret & str
            parent.AppendChild(add)
        End If
    Next
    Return ret
End Function
Public Function getSHA1Hash(ByVal strToHash As String) As String
    Dim sha1Obj As New System.Security.Cryptography.SHA1CryptoServiceProvider
    Dim bytesToHash() As Byte = System.Text.Encoding.ASCII.GetBytes(strToHash)
    bytesToHash = sha1Obj.ComputeHash(bytesToHash)
    Dim strResult As String = ""
    For Each b As Byte In bytesToHash
        strResult += b.ToString("x1")
    Next
    Return strResult
End Function
Public Function secstruct(ByVal list As System.Xml.XmlNodeList, ByVal parent As
System.Xml.XmlElement) As Integer
    Dim att As System.Xml.XmlAttributeCollection
    Dim attr As System.Xml.XmlAttribute
    Dim node As System.Xml.XmlNode
    Dim add As System.Xml.XmlElement
    For Each node In list
        If node.NodeType = System.Xml.XmlNodeType.Element Then
            att = node.Attributes
            For i = 0 To att.Count - 1
                attr = att.ItemOf((att.Count - 1) - i)
                If attr.Name = "hash" Then
                    add = xdwr.CreateElement("x" & attr.Value)
                End If
```

```
            If attr.Name = "policy" Then
              add.SetAttribute("policy", attr.Value)
            End If
          Next
          parent.AppendChild(add)
          If node.HasChildNodes And node.ChildNodes.Count > 1 Then
            secstruct(node.ChildNodes, add)
          End If
        End If
      Next
      Return 0
    End Function
    Protected Sub Button1_Click(ByVal sender As Object, ByVal e As
System.EventArgs) Handles Button1.Click
      Dim xdoc As System.Xml.XmlDocument = New System.Xml.XmlDocument()
      Dim se, tr As String
      xdoc.Load("C:\Inetpub\wwwroot\owner1\ownerdoc\xmlpol.xml")
      xdwr.AppendChild(xdwr.CreateElement(xdoc.DocumentElement.Name))
      se = hashcode(xdoc.DocumentElement.ChildNodes, xdwr.DocumentElement)
      xdwr.DocumentElement.SetAttribute("max",
xdoc.DocumentElement.GetAttribute("max"))
      tr = getSHA1Hash(xdwr.DocumentElement.Name)
      xdwr.DocumentElement.SetAttribute("own", tr)
      xdwr.DocumentElement.SetAttribute("hash", getSHA1Hash(tr & se))
      xdwr.Save("C:\Inetpub\wwwroot\owner1\ownerdoc\xmlhash.xml")
      ifrm.Attributes.Remove("src")
      ifrm.Attributes.Add("src", "http://localhost/owner1/ownerdoc/xmlhash.xml")
    End Sub
  Protected Sub Button2_Click(ByVal sender As Object, ByVal e As System.EventArgs)
  Handles Button2.Click
      Dim rsa As New System.Security.Cryptography.RSACryptoServiceProvider(1024)
```

```
        Dim str, sign, cypher As String
        Dim xpath As System.Xml.XPath.XPathNavigator
        Dim xnode As System.Xml.XPath.XPathNodeIterator
        Dim inp(), oup() As Byte
        Dim pub As System.Xml.XmlDocument = New System.Xml.XmlDocument()
        Dim sr As System.IO.StreamReader = New
System.IO.StreamReader("C:\Inetpub\wwwroot\owner1\ownerdoc\PrivateKey.xml")
        str = sr.ReadToEnd()
        sr.Close()
        rsa.Clear()
        rsa.FromXmlString(str)
        pub.Load("C:\Inetpub\wwwroot\owner1\ownerdoc\xmlhash.xml")
        str = pub.DocumentElement.GetAttribute("hash")
        inp = System.Text.Encoding.Unicode.GetBytes(str)
        oup = rsa.Encrypt(inp, False)
        cypher = Convert.ToBase64String(oup)
        xpath = pub.CreateNavigator()
        xnode = xpath.Select("/" & pub.DocumentElement.Name)
        xnode.MoveNext()
        xnode.Current.CreateAttribute("fdf", "sign", "", cypher)
        pub.Save("C:\Inetpub\wwwroot\owner1\ownerdoc\xmlhashsign.xml")
        ifrm.Attributes.Remove("src")
        ifrm.Attributes.Add("src", "http://localhost/owner1/ownerdoc/xmlhashsign.xml")
    End Sub
    Protected Sub Button3_Click(ByVal sender As Object, ByVal e As
System.EventArgs) Handles Button3.Click
        Dim read As New System.Xml.XmlDocument
        read.Load("C:\Inetpub\wwwroot\owner1\ownerdoc\xmlhashsign.xml")
        xdwr.AppendChild(xdwr.CreateElement("secstruct"))
        xdwr.DocumentElement.SetAttribute("max",
read.DocumentElement.GetAttribute("max"))
```

```
    secstruct(read.ChildNodes, xdwr.DocumentElement)
    xdwr.Save("C:\Inetpub\wwwroot\owner1\ownerdoc\secstruct.xml")
    ifrm.Attributes.Remove("src")
    ifrm.Attributes.Add("src", "http://localhost/owner1/ownerdoc/secstruct.xml")
  End Sub
End Class
```

6.1.2 Subject Hash Verification:

Subject Hash Verification.aspx.vb:

```
Partial Class verhash
  Inherits System.Web.UI.Page
  Public Function verisign(ByVal repdoc As System.Xml.XmlDocument) As String
    Dim str, sign As String
    Dim nodelist As System.Xml.XmlNodeList
    Dim node As System.Xml.XmlNode
    Dim sr As System.IO.StreamReader = New
System.IO.StreamReader("C:\Inetpub\wwwroot\publisher\pubdoc\PublicKey.xml")
    str = sr.ReadToEnd()
    sr.Close()
    Dim rsa As New System.Security.Cryptography.RSACryptoServiceProvider(1024)
    rsa.FromXmlString(str)
    Dim inp(), oup() As Byte
    str = repdoc.DocumentElement.GetAttribute("sign")
    inp = Convert.FromBase64String(str)
    oup = rsa.Decrypt(inp, False)
    sign = System.Text.Encoding.Unicode.GetString(oup)
    nodelist = repdoc.DocumentElement.ChildNodes
    Label2.Text = sign
    str = ""
    Dim poi As Integer
    poi = 0
```

```
    For Each node In nodelist
        str = computehash(node)
        If str <> sign Then
            Label6.Text = "Not Equal"
            Label8.Text = "Data Modified"
            Label4.Text = str
            poi = 1
        End If
    Next
    If poi = 0 Then
        Label4.Text = str
        Label6.Text = "Equal"
        Label8.Text = "Data Not Modified"
    End If
    Return str
End Function
Public Function getSHA1Hash(ByVal strToHash As String) As String
    Dim sha1Obj As New System.Security.Cryptography.SHA1CryptoServiceProvider
    Dim bytesToHash() As Byte = System.Text.Encoding.ASCII.GetBytes(strToHash)
    bytesToHash = sha1Obj.ComputeHash(bytesToHash)
    Dim strResult As String = ""
    For Each b As Byte In bytesToHash
        strResult += b.ToString("x1")
    Next
    Return strResult
End Function
Public Function computehash(ByVal enode As System.Xml.XmlNode) As String
    Dim node As System.Xml.XmlNode
    Dim sa, own, attstr, sing, has, chech As String
    Dim attrs As System.Xml.XmlAttributeCollection
    Dim attr As System.Xml.XmlAttribute
```

```
    Dim nodelist As System.Xml.XmlNodeList
    Dim nav As System.Xml.XPath.XPathNavigator
    sa = ""
    nodelist = enode.ChildNodes
    For Each node In nodelist
      If node.NodeType = System.Xml.XmlNodeType.Element Then
        If node.Name = "hash" Then
          sa = sa & node.InnerText
        ElseIf node.Name = "own" Then
          attrs = node.Attributes
          For Each attr In attrs
            If attr.Name = "own" Then
              own = attr.Value
            End If
          Next
          chech = computehash(node)
          sa = sa & getSHA1Hash(own & chech)
        Else
          attrs = node.Attributes
          attstr = ""
          For Each attr In attrs
            If attr.Name <> "hash" And attr.Name <> "own" And attr.Name <>
"verhash" And attr.Name <> "policy" Then
              attstr = attstr & getSHA1Hash(getSHA1Hash(attr.Name) &
getSHA1Hash(attr.Value))
            End If
          Next
          has = getSHA1Hash(node.Name)
          sing = ""
          If node.HasChildNodes And node.ChildNodes.Count > 1 Then
            sing = datahash(node)
```

```
            Else
              has = getSHA1Hash(has & getSHA1Hash(node.InnerText))
            End If
            has = getSHA1Hash(has & attstr)
            If node.HasChildNodes And node.ChildNodes.Count > 1 Then
              sing = getSHA1Hash(has & sing)
            Else
              sing = has
            End If
            nav = node.CreateNavigator()
            attrs = node.Attributes
            Dim atr As System.Xml.XmlAttribute
            Dim poi As Integer
            poi = 0
            For Each attr In attrs
              If attr.Name = "verhash" Then
                atr = attr
                poi = 1
              End If
            Next
            If poi = 1 Then
              attrs.Remove(atr)
            End If
                  nav.CreateAttribute("", "verhash", "", sing)
            sa = sa & sing
          End If
        End If
      Next
      Return sa
End Function
Public Function datahash(ByVal enode As System.Xml.XmlNode) As String
```

```
    Dim str, attstr, ret, one As String
    Dim att As System.Xml.XmlAttributeCollection
    Dim attr As System.Xml.XmlAttribute
    Dim node As System.Xml.XmlNode
    ret = ""
    Dim list As System.Xml.XmlNodeList
    list = enode.ChildNodes
    For Each node In List
      If node.NodeType = System.Xml.XmlNodeType.Element Then
        If node.HasChildNodes = True And node.ChildNodes.Count > 1 Then
          one = datahash(node)
          att = node.Attributes
          attstr = ""
          For Each attr In att
            If attr.Name <> "hash" And attr.Name <> "own" And attr.Name <>
"verhash" And attr.Name <> "policy" Then
              attstr = attstr & getSHA1Hash(getSHA1Hash(attr.Name) &
getSHA1Hash(attr.Value))
            End If
          Next
          str = getSHA1Hash(node.Name)
          str = getSHA1Hash(str & attstr)
          str = getSHA1Hash(str & one)
        Else
          att = node.Attributes
          attstr = ""
          For Each attr In att
            If attr.Name <> "hash" And attr.Name <> "own" And attr.Name <>
"verhash" And attr.Name <> "policy" Then
              attstr = attstr & getSHA1Hash(getSHA1Hash(attr.Name) &
getSHA1Hash(attr.Value))
```

```
            End If
          Next
          If node.Name <> Nothing Then
            str = getSHA1Hash(node.Name)
          End If
          If node.InnerText <> Nothing Then
            str = getSHA1Hash(str & getSHA1Hash(node.InnerText))
          End If
          str = getSHA1Hash(str & attstr)
        End If
        ret = ret & str
      End If
    Next
    Return ret
  End Function
Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
Handles Me.Load
    Dim asd As New System.Xml.XmlDocument
    asd.Load("C:\Inetpub\wwwroot\publisher\pubdoc\reply.xml")
    verisign(asd)
    asd.Save("C:\Inetpub\wwwroot\publisher\pubdoc\reply.xml")
  End Sub
End Class
```

6.1.3 Subject Completeness Verification:

Subject Completeness Verification.aspx.vb:

```
Partial Class complete
  Inherits System.Web.UI.Page
  Dim max As Integer
  Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
Handles Me.Load
```

```
        Dim avail, lblptr, pubptr As Integer
        Dim pol(8) As Integer
        Dim sp(8) As String
        Dim ty, nonodes As Integer
        Dim lab(20) As System.Web.UI.WebControls.Label
        Dim labpub(20) As Label
        Dim reslab(20) As Label
        Dim reply As System.Xml.XmlTextReader = New
System.Xml.XmlTextReader("C:\Inetpub\wwwroot\publisher\pubdoc\reply.xml")
        reply.Read()
        reply.MoveToAttribute("polstr")
        sp = Split(reply.Value, ",")
        ty = 0
        nonodes = 0
        For i = 0 To 7
          If sp(i) = "" Then
            pol(i) = 0
          Else
            pol(i) = Convert.ToInt32(sp(i))
          End If
        Next
        lblptr = 0
        While pol(ty) <> 0
          Dim repver As System.Xml.XmlTextReader = New
System.Xml.XmlTextReader("C:\Inetpub\wwwroot\publisher\pubdoc\secstruct.xml")
          repver.Read()
          repver.MoveToAttribute("max")
          max = repver.Value
          Dim ref As Integer
          If max Mod 4 = 0 Then
            max = max
```

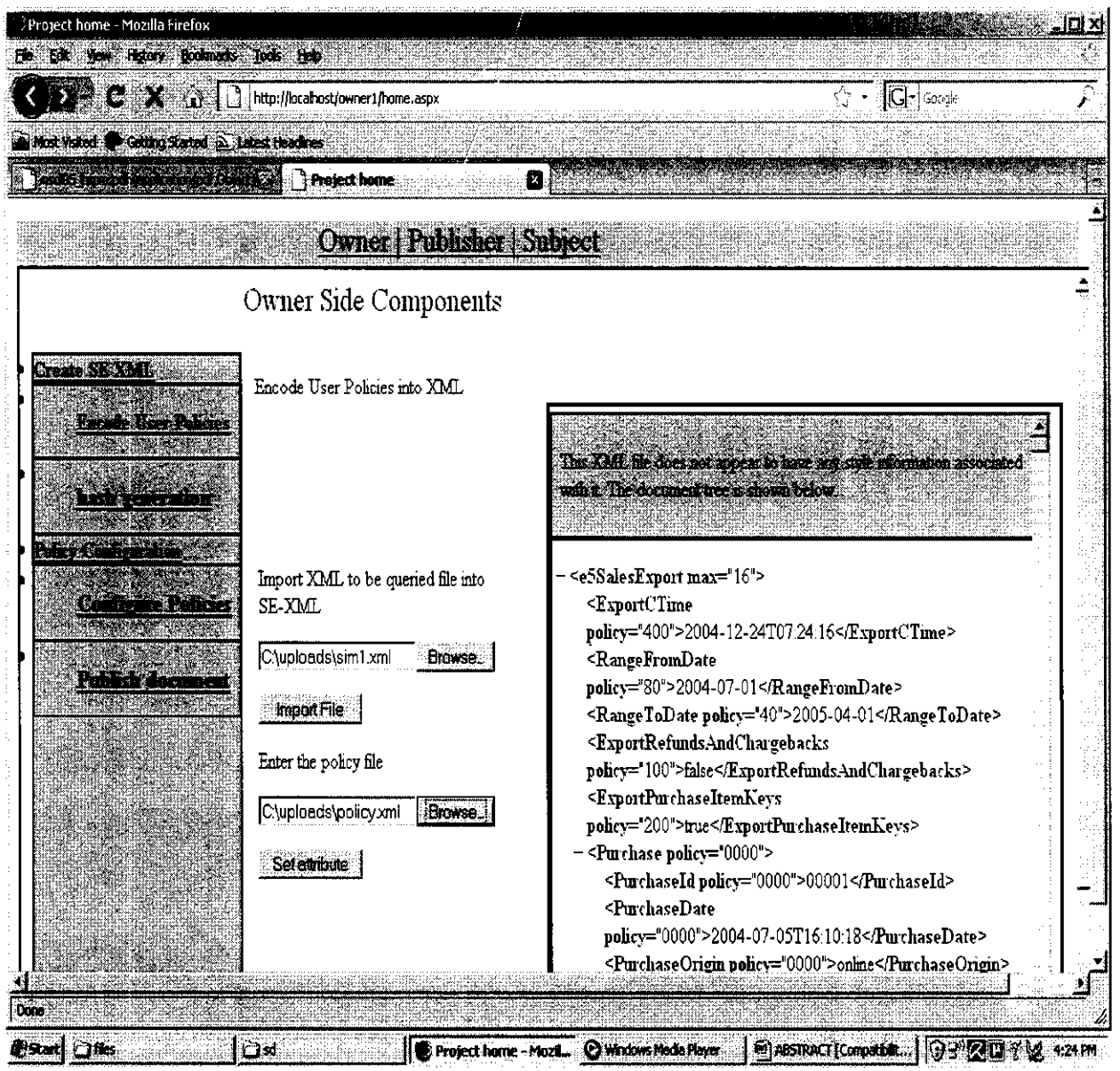
```
        Else
          ref = max / 4
          max = (ref + 1) * 4
        End If
        While repver.Read()
          If repver.MoveToAttribute("policy") Then
            avail = Integer.Parse(repver.Value,
Globalization.NumberStyles.AllowHexSpecifier)
            Dim sd As String = Convert.ToString(avail, 2)
            Dim bit As New BitArray(sd.Length)
            Dim dup As New BitArray(max)
            For i = 0 To bit.Length - 1
              bit(i) = (sd.Chars(i) = "1"c)
            Next
            Dim dr, sr As Integer
            dr = dup.Length - bit.Length
            sr = dr
            For i = 0 To dup.Length - 1
              If dr > 0 Then
                dup(i) = False
                dr = dr - 1
              Else
                dup(i) = bit(i - sr)
              End If
            Next
            If dup.Length > pol(ty) - 1 Then
              If dup(pol(ty) - 1) = True Then
                lab(lblptr) = New Label
                repver.MoveToElement()
                Dim str1 As String
                str1 = repver.Name
```

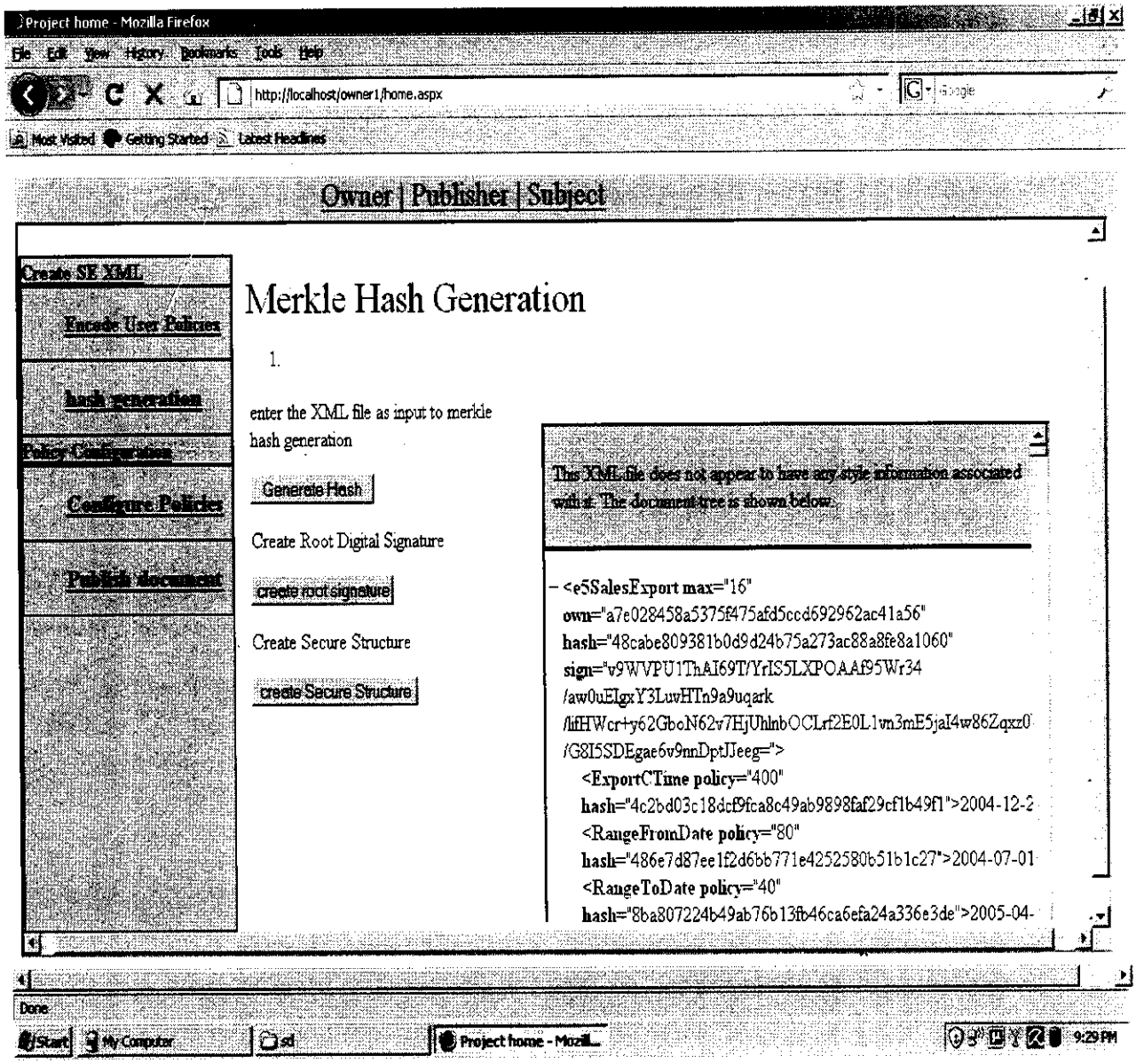
```
                If Not repver.IsEmptyElement() Then
                   repver.Read()
                    While repver.Name <> str1
                      repver.Read()
                    End While
                End If
                str1 = str1.Substring(1)
                lab(lblptr).Text = "<br />" & str1
                div1.Controls.Add(lab(lblptr))
                lblptr = lblptr + 1
            End If
          End If
        End If
      End While
      ty = ty + 1
      repver.Close()
End While
Label1.Text = lblptr
pubptr = 0
While reply.Read()
   If reply.MoveToAttribute("verhash") Then
      labpub(pubptr) = New Label
      labpub(pubptr).Text = "<br />" & reply.Value
      div2.Controls.Add(labpub(pubptr))
      pubptr = pubptr + 1
   End If
End While
Label2.Text = pubptr
If lblptr = pubptr Then
   For i = 0 To lblptr - 1
      reslab(i) = New Label
```

```
            If labpub(i).Text = lab(i).Text Then
               reslab(i).Text = "<br />complete"
            Else
               reslab(i).Text = "<br />Incomplete"
            End If
            div3.Controls.Add(reslab(i))
         Next
      Else
         Label3.Text = "publisher has not returned correct nodes"
      End If
      reply.Close()
   End Sub
End Class
```

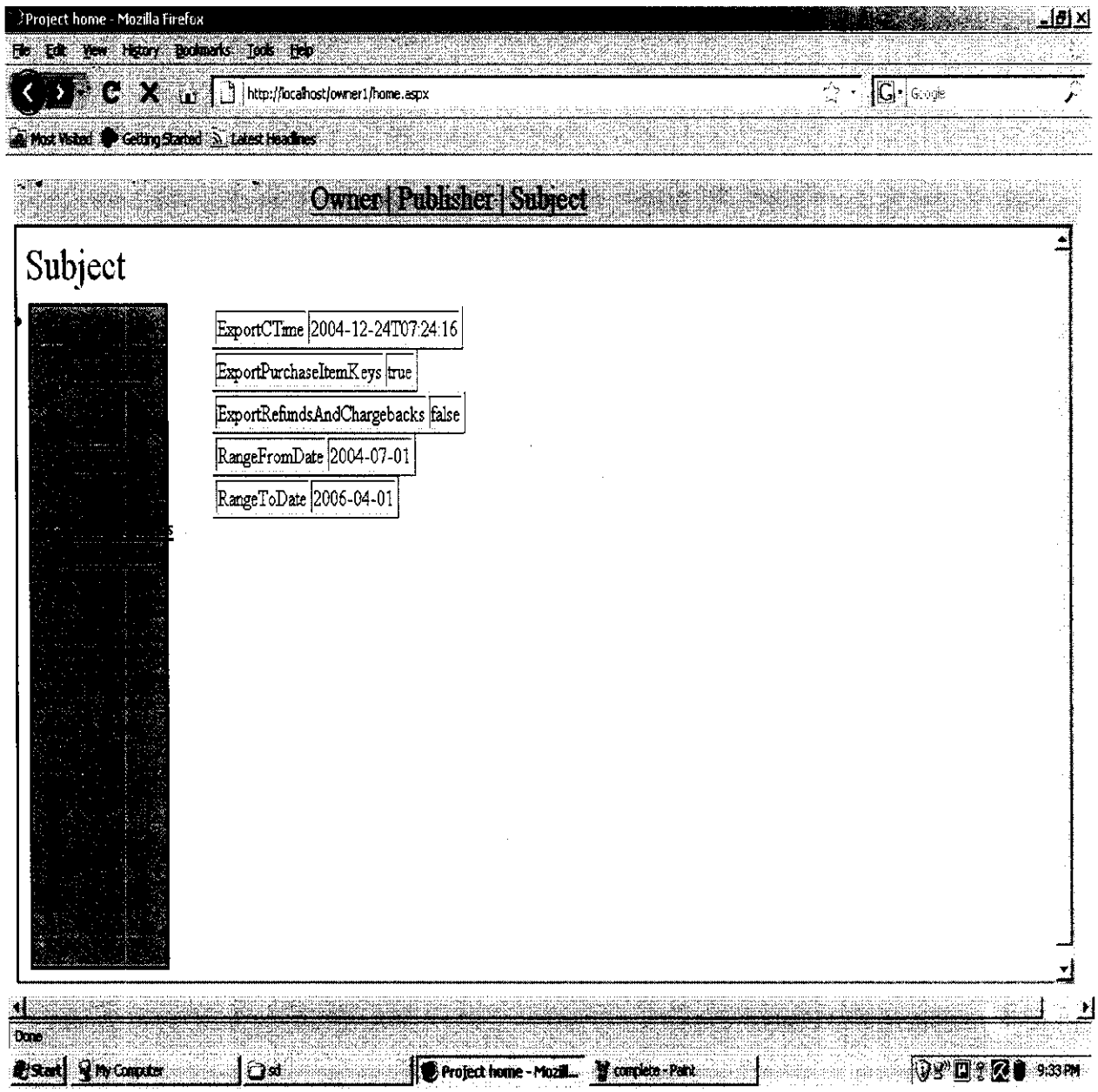
6.2 ScreenShots:

6.2.1 Create SE-XML:

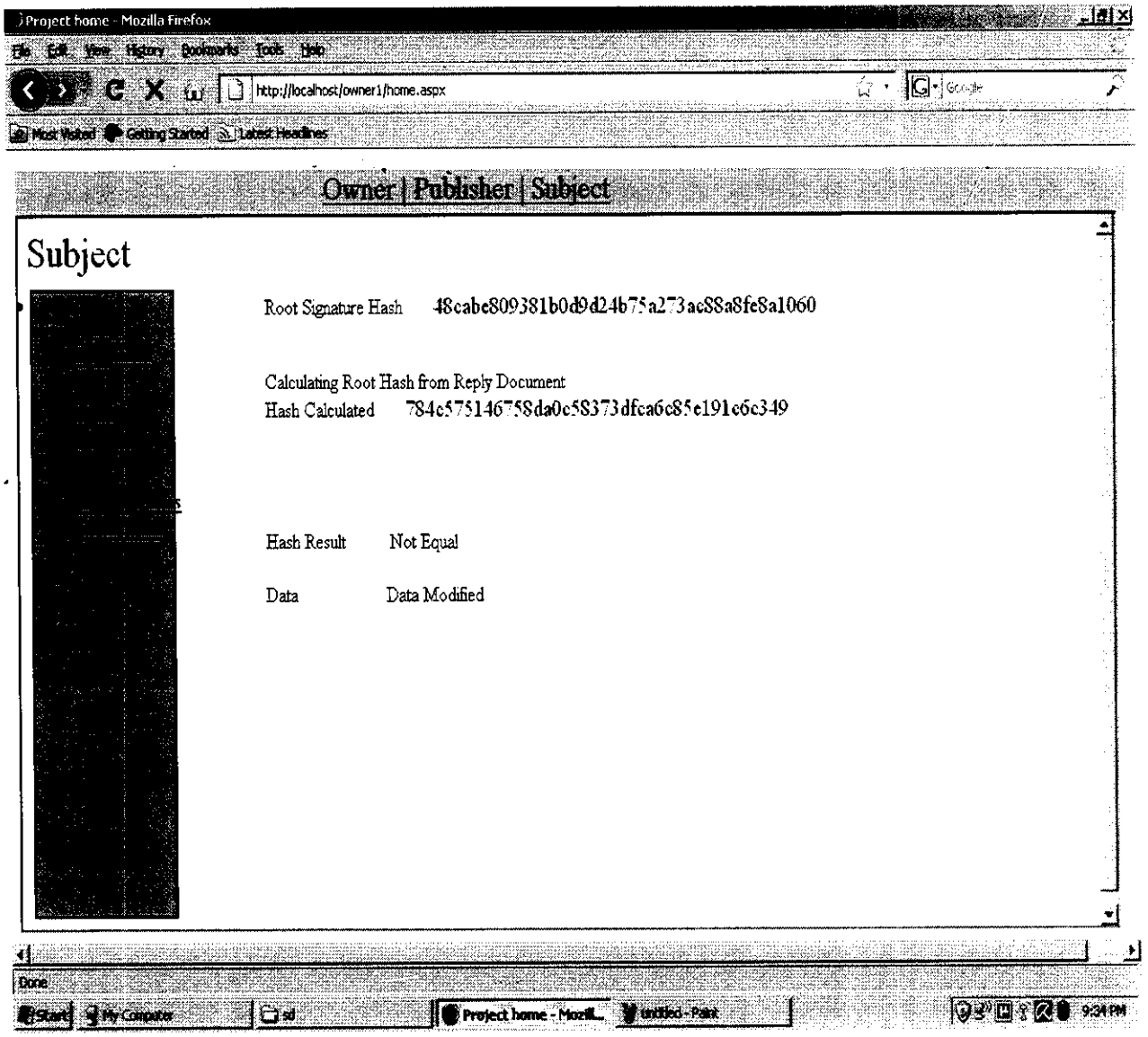



Project home - Mozilla Firefox
http://localhost/owner1/home.aspx
Owner | Publisher | Subject
Create SE XML
Encode User Policies
hash generation
Policy Configuration
Configure Policies
Publish document
Merkle Hash Generation
1.
enter the XML file as input to merkle hash generation
Generate Hash
Create Root Digital Signature
create root signature
Create Secure Structure
create Secure Structure
This XML file does not appear to have any style information associated with it. The document tree is shown below.
- <e5SalesExport max="16"
own="a7e028458a5375f475afd5ccd692962ac41a56"
hash="48cabe809381b0d9d24b75a273ac88a8fe8a1060"
sign="v9WVPU1ThAI69T/YrIS5LXPOAAf95Wr34
/aw0uElgxY3LuvHTn9a9uqark
/lifHWcr+y62GboN62v7HjUhlnbOCLrf2E0L1vn3mE5jaI4w86Zqxz0
/G8I5SDEgae6v9nnDptJJeeg=">
<ExportCTime policy="400"
hash="4c2bd03c18dcf9fca8c49ab9898faf29cf1b49f1">2004-12-2
<RangeFromDate policy="80"
hash="486e7d87ee1f2d6bb771e4252580b51b1c27">2004-07-01
<RangeToDate policy="40"
hash="8ba807224b49ab76b13fb46ca6efa24a336e3de">2005-04-
Done
Start
My Computer
sd
Project home - Mozil...
9:29 PM

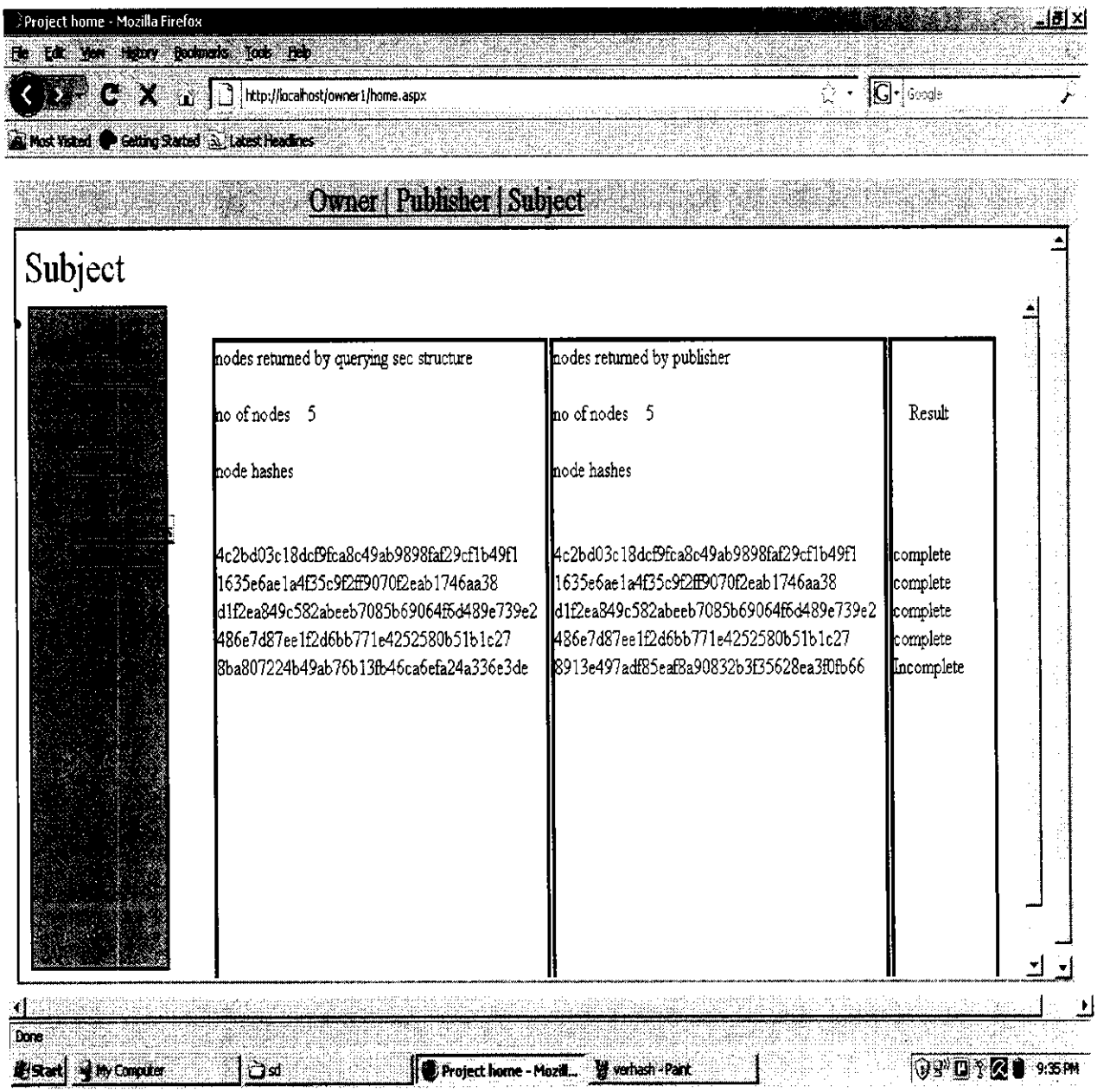
6.2.2 Reply Generation:



6.2.3 Subject Hash Verification:



6.2.4 Subject Completeness Verification:



REFERENCES

8.REFERENCES:

[1] E. Bertino, B. Carminati, and E. Ferrari, "A Temporal Key Management Scheme for Secure Broadcasting of XML Documents,"Proc. Ninth ACM Conf. Computer and Comm. Security, pp. 31-40, 2002.

[2] E. Bertino, S. Castano, and E. Ferrari, "On Specifying Security Policies for Web Documents with an XML-Based Language," Proc.Sixth ACM Symp. Access Control Models and Technologies, pp. 57-65, 2001.

[3] E. Bertino, S. Castano, and E. Ferrari, "Authorx: A Comprehensive System for Securing XML Documents," IEEE Internet Computing,vol. 5, no. 3, pp. 21-31, May/June 2001.

[4] L. Cranor and J. Reagle, "The Platform for Privacy Preferences," Comm. ACM, vol. 42, no. 2, pp. 48-55, 1999.

Web References:

[1] World Wide Web Consortium, Xml, http://www.w3.org/XML,2004.

[2]Merkle hash trees,www.rsa.com/rsalabs/node.asp?id=2003

[3]XML,Xpath,www.w3.org/TR/xpath20/

[4] Vb.net tutorials, http://sourceforge.net/