CHAPTER 1

INTRODUCTION

1.1 INTRODUCTION

The eXtensible Markup Language (XML) is a universal format for structured documents and data over the web. The Hypertext Markup Language (HTML) format describes only the format of the data, and therefore, programs cannot understand the structure of such data. On the other hand, XML provides flexibility in the representation of the structured documents and data.

The World Wide Web Consortium (W3C) introduced XML to complement HTML, for data exchange over the web. Bray et al (2008) explained the most recent version of the XML language recommendation. XML is a powerful tool, meant to deal with the current structure of document information processing. It is a cross-platform Internet environment technology, which is dependent on the content. XML documents (marked-up documents) are self-describing. Therefore, it is possible for programs to interpret the meaning of the data, which are received, filter the document based on its content, and restructure it to suit the application's needs. XML documents provide a platform independent means to describe data, and hence, they can transport data from one platform to another.

XML is a simplified subset of the Standard Generalized Markup Language (SGML), the international standard meta-language for text markup
systems. XML is designed to improve the functionality of the web, by providing more flexible and adaptable information identification. It is called extensible because it allows the user to create his/her own tags that describe the data, unlike HTML that has pre-defined tags to format the data on the web. Although XML takes more space than the binary data, XML is extremely simple and easy to master and use. These features make XML the first choice for the exchange of information between the Simple Object Access Protocol (SOAP) and web services.

Since XML is structured data, it is suitable for a structured data transmission network. Therefore, in recent years, XML applications have been growing. Security has always been of importance to ensure data protection, transactions' integrity, and maintain information privacy and confidentiality. One of the most challenging problems in managing large, distributed, and heterogeneous networked systems, is the ability to specify and enforce security policies between parties, and ensure access to services and resources. The infrastructure over the Internet has turned out to be difficult, due to system heterogeneity and conflicting security requirements.

1.2 SECURITY FOR XML DATA

Nowadays, the main requirement of Internet-wide security standards, is applying the security to the content created, using XML. XML has been adopted widely for a great variety of applications and types of content. XML is used to apply access control on the structure and content of a document. Moreover, XML is the basis of web services protocols that rely on different XML-based languages, such as SOAP, Web Service Definition Language (WSDL), and Universal Description Discovery and Integration (UDDI). Therefore, it is important to enforce the security of XML documents, to realize the exchange of XML documents in anonymous and untrustworthy
environments, and to ensure confidentiality, integrity, authenticity, and non-repudiation of XML documents (Ardagna et al 2007).

The existing security technologies provide a set of security technologies used in XML security. However, the formats used to satisfy the security requirements are not compatible with the XML security applications. One reason is that, the existing security technologies use binary formats that require specific tools for interpretation and use. Another reason is that, these security technologies are not designed for XML. The XML security system addresses these issues, by reusing the concepts, algorithms, and core technologies of legacy security systems, while introducing changes necessary to support an extensible integration with XML. The XML security standards are as follows (Hirsch 2002):

1. XML Digital Signature for integrity and signing solutions (Bartel et al 2013).
3. XML Key Management Specification (XKMS) for public key registration, location, and validation (Hallam-Baker & Mysore 2005).
5. EXtensible Access Control Markup Language (XACML) for describing policies and defining access control rules (Rissanen 2013).
Compared to traditional security technologies, there are four major advantages related to XML security.

1. The ability to selectively encrypt and sign portions of a message.
2. The ability to protect data integrity without encrypting it.
3. The ability to construct overlapping digital signatures using different keys.
4. The ability to digitally sign and encrypt data.

The encryption and signature specifications proposed by W3C specifying the format for encrypted XML documents, are important to XML security. However, they cannot allow the programmer to specify how to encrypt and sign the XML documents. Existing XML security specifications have some problems, which need to be improved, such as the efficiency of the authentication process, context-referral integrity for XML data in XML signature, multi-signature generation, and encrypted XML data query.

Moreover, although XACML is an integral policy description language, the structure of the existing XACML policy is very complex and hard. Hence, it is necessary for the users to understand XACML well, in order to write all the securing policy specifications. On the other hand, the Query languages of a RDBMS are easy and simple to use by all the users. Moreover, SQL-like query languages overcome the difficulties of XACML, by storing the XACML policies and rules in relations. Therefore, it is easy for the users to use and understand the XACML policies and rules. Moreover, storing the data and rules in tables provides more flexibility. Hence, it is necessary to provide the table based storage features of web databases.
Most of the researches are aimed towards representation and querying, authorization, and access control policies, for safeguarding the XML information system stored at the server. Therefore, it is necessary to provide a new algorithm for providing security for temporal XML based databases, not only for the effective representation of the temporal XML document, but also for providing facilities for encryption and decryption.

1.3 OBJECTIVES OF THE RESEARCH WORK

The main objectives of this research work are as follows:

1. To propose new algorithms for overcoming the complex structure of the XACML policies and rules by secure mapping of the XACML policies and rules into temporal relations.

2. To develop a secure temporal database based on XML.

3. To propose a new XML based storage structure for databases.

4. To provide the capability for decrypting the encrypted XML documents partially.

1.4 THESIS CONTRIBUTIONS

This research work aims to overcome the complex structure of the XACML policies and rules by secure mapping of them into temporal relations. Moreover, it aims to provide the capability for decrypting the encrypted XML documents partially. To fulfil these objectives, the thesis provides the following contributions:
1. Proposes new algorithms for mapping and storing the XACML policies and rules in the form of rules in temporal relations.

2. Proposes a new time-stamp based algorithm, for decrypting the encrypted XML documents partially.

The first contribution is proposing new algorithms for mapping the XACML policies and rules into relational rules, and storing them in the form of rules in temporal relations, to ease the access control to the XML documents. This proposed work relieves the users from the effort of learning and understanding the XACML policies and rules; hence, it saves the users' time and effort. It controls the access to the XML documents stored in either native or relational databases, using the XACML policies. Finally, it applies the constraints of rules, and obligations and provides the response to an access request effectively.

The second contribution of this research work is proposing a new time-stamp based algorithm, for decrypting the encrypted XML documents partially. This proposed algorithm utilizes a new temporal model for XML representation, based on a combination of valid time and transaction time, through the tuple time stamping to decrypt parts of the encrypted XML documents. Moreover, one attribute of the key components of the encrypted elements is selected to distinguish among them. The time-stamp and the selected attribute guide the receiver to decrypt the required parts of the encrypted XML documents, instead of decrypting all the parts of them. The proposed algorithm not only reduces the decryption time, but also protects the security of the data.

The new temporal model is proposed not only for the effective representation of the temporal XML documents, but also for providing
facilities for encryption and decryption; i.e., the receiver uses the time-stamp and the selected attribute to retrieve the parts that must be decrypted, to contribute to the query results on the encrypted XML documents. Moreover, this model uses the valid time and transaction time for time-stamping, and hence, can maintain the history data efficiently. This model can be used to develop secure temporal applications

1.5 OUTLINE OF THE THESIS

The outline of this thesis is as follows:

1. **Chapter 2** presents a survey of the recent research that described and implemented the XML security standards and access control to the XML documents.

2. **Chapter 3** shows the background information about the XML signature and encryption.

3. **Chapter 4** depicts the standard XACML and the Role-Based Access Control (RBAC) profile of XACML.

4. **Chapter 5** explains the new algorithms for mapping the XACML policies and rules into a RDBMS.

5. **Chapter 6** discusses the time-stamp based algorithm, for decrypting the encrypted XML documents partially.

6. **Chapter 7** gives the conclusions of this work and suggests the future work.
CHAPTER 2

LITERATURE REVIEW

In the past, many research works have been carried out on XML security and access control to the XML documents. This chapter provides a survey of the related work in these areas.

2.1 XML SECURITY


Schadow (2005) explained the use of the XML-Security Plug-In for Eclipse and CrypTool, which allow the users to apply XML encryption and signature to XML documents effectively. Schrefl et al (2005) presented a security method to ensure the privacy of documents, which are stored in the server after client side encryption.

Lee & Whang (2006) proposed a new Query Aware Decryption algorithm for the effective processing of XML queries against encrypted XML data. Their algorithm allows the users to decrypt not only the parts of
the query results but also full query results. Moreover, they used an XML index along with the encrypted XML data for enhancing the processing performance since this index helps to specify the location of the query results in the encrypted XML data. Yang et al (2006) proposed a Queriable XML Encryption (XQEnc) method, which is a new XML encryption technique, based on XML vectorization and used skeleton compression techniques. Their system computes compressed skeleton and a set of data vectors, which are encrypted separately. It also includes SAML and XACML.

Vasudevan & Yang (2006) proposed a rule-based security engine that is capable for enforcing security policies by enhancing the Apache Axis. By using this engine, it is possible for the developers of web services to declare the necessary security attributes for securing their applications. The main advantage of their approach is that, it allows reusing the security engine for different web services. Weerasinghe et al (2006) presented an XML encryption and signature algorithm based on mobile web services environment for health care application to protect data against malicious users.

Ardagna et al (2007) presented about XML signature and encryption, for providing security to XML databases. Moreover, they explained XKMS, which provides the registration, location, and validation techniques for public keys, used together with XML signature and encryption. They also presented two XML-based access control languages, namely, XACML and WS-Policy. Kato et al (2007) applied XML signature and encryption to provide authentication, data confidentiality, and integrity in a peer-to-peer platform. Kangasharju (2007) presented an XML encryption and signature technique, which can be used in mobile devices to provide security to mobile devices. Yixiang et al (2007) explained about information leakage while publishing XML documents and they proposed a new algorithm, called
Eliminate Inner Nodes (EIN) algorithm, in order to protect the sensitive data. Moreover, their algorithm is capable for retrieving a partial document from an XML document securely without causing information leakage.

Chang & Hwang (2007) proposed a new language, called Document Security Language (DSL), which supports the operational model of the document security system. Qiao (2007) demonstrated that the XML united signature structure could resolve the authenticity, integrity and non-repudiation problems of the data in the multi-operation chain, based on the XML communication. In their work, the <UnitedSignature> is the root element of the XML united signature, and it contains three elements, namely, <SubDigests>, <Signature>, and <UnitedSignatureObject>. Cho (2007) proposed an integration between XML encryption and access control to achieve the security requirements in the level of transport layer and in the access of different users.

Hai-hua et al (2008) proposed a new digital signature technology, since the existing signature schemas are not sufficient to handle the new XML requirements of a fine-grained signature and multiple signatures. Their proposed multi-signature scheme is based on the Ron Rivest, Adi Shamir, and Leonard Adleman (RSA) signature algorithm. Their model uses the XPath transform rules of the XML correlation techniques to divide the documents into sub-documents. In this schema, the owner digitally signs each sub-document, which can be verified by the receiving entities. Kundu & Bertino (2008) proposed a new approach for content dissemination, which has the structural properties of the XML DOM. Their approach provides an efficient dissemination and assures both content integrity and confidentiality. This is due to the fact that, it based on the encrypted postorder numbers, which supports the integrity and confidentiality requirements of XML content. Chang & Hwang (2008) proposed a new XML query language,
called the secure XML Query (sXQuery) language with an editor based on the XQuery language. sXQuery provides the features of both XQuery and DSL languages.

U¨nay & Gu¨ndem (2008) presented a survey on indexes used for querying the encrypted XML documents. Their survey explained two types of indexes, namely, structural and value indexes and presented the various techniques used in indexing at both client side and server side. The structural index determines whether the path in the XML query matches with any of the paths in the XML documents. The value index checks the constraints provided in the range queries. Moreover, it lists the possible attack types and the cryptanalysis techniques to be used for encrypted XML documents. Sun & Li (2008) explained about XML and web services security standards in their paper. The XML security standards include the XML digital signature and encryption.

Gao et al (2008) proposed the XFlat technique for access control to XML documents. This technique is more concerned on query performance on the published XML view and thus protects the sensitive data using encryption techniques. By using an XML index tree, Xi-quan et al (2008) proposed a transformation-based algorithm for the retrieved of information based on semantic path. Their proposed algorithm is useful to the users to find subtle granular and helps in significant information to be signed quickly. Though their proposed algorithm enhanced the efficiency of XML multi-signature, it did not affect the data security. Brechlerova & Candik (2008) demonstrated the technologies of XML security and described their benefits in health documentation.

used for implementing the dynamic data handling on a web browser. Hashizume & Fernandez (2009) presented two patterns, namely, a Symmetric Encryption pattern to describe the basic type of algorithms and the other pattern is the XML Encryption, which describes a method of applying symmetric and asymmetric encryption techniques to XML messages. Knap & Mlˇynkov´a (2009a) analyzed the current security challenges in the XML signature and presented the suitable solutions.

Nordbotten (2009) presented about the XML security standards and web services. The XML security standards include XML signature, XML encryption, XACML, SAML, and XKMS. The web services include WS-Security, WS-Trust, WS-SecureConversation, Web Services Policy, and WS-SecurityPolicy. Jensen et al (2009) presented about XML namespaces in the domain of XML signatures since it has critical deficiencies, which can lead to vulnerabilities through XML signature wrapping attacks. Moreover, they described the problem of namespace injection and showed an attack scenario based on this technique. Moreover, they discussed several new approaches to overcome this threat.

Doroodchi et al (2009) presented about the service security based on XML and the various forms of XML-based attacks. Moreover, they provided recommendations and countermeasures for the attacks. Ping & Laihong (2009) presented a new approach on how to use the XML encryption and signature for effective logistics data exchange. Elgedawy et al (2009) proposed a new query-aware approach for compressing and encrypting large XML documents, while maintaining queriability over the intermediate document. Their proposed approach separates the document structure from its contents using the Ctree+ XML indexing approach. In addition to the separation, it applies context-free lossless encryption and compression techniques over the Ctree+ intermediate representations. Knap & Mlˇynkov´a (2009b) presented different processing approaches to the process of
verification of the web services integrity. Rahaman et al (2009) proposed an ontology-based XML content distribution system to protect the document content from unauthorized users and to protect the document structure form other organizations.

Xuan-min et al (2010) proposed to retrieve the encrypted XML data, using value and structure indexes, and discussed about the bucket management of the entrance addresses. Ammari & Lu (2010) proposed a new architecture to handle bulk XML messages, which has the ability to encrypt the sensitive parts of each message effectively using different types of encryption. Jie (2010) proposed an algorithm for XML document information management, based on equal element method. Their algorithm is capable of transforming a secret message into a decimal integer and creates an equal element by applying permutations and combinations to sub-elements. It inserts an integer into the XML document by exchanging some elements with their equal element based on a mapping function defined from the equal element to the integer. Jing (2010) analyzed about security system and the role of the XML technology in the security significance.

Zhihong & Yu (2010) presented a new system in university information platform. The system uses XML encryption and signature algorithm. They considered authentication, confidentiality, and integrity, while performing transfer and storage of the data. Yan & Xiuping (2010) proposed a system using XML encryption to provide a secure transmission of electronic records pertaining to medical applications. Chen et al (2010) presented a model with grammar structure, implementation, and its application for security based on XML digital signature.

Al-Hamdani (2010) proposed the use of XML for security implementation in their web-based healthcare applications. Luo et al (2010) implemented the Attribute Based Encryption (ABE) technique in web
services. Their technique aims to provide effective security and privacy preservation mechanism in web databases, instead of using XML encryption and XACML. Almarimi & Alsahdi (2010) proposed a cryptosystem for the encryption as well as the decryption of XML documents using a hybrid of RSA and Shift cipher algorithms. Their system improves the security by enhancing confidentiality, authentication, integrity, and non-repudiation. Haron et al (2010) proposed a Document Management System (DMS) for a web environment based on C/C++. Their DMS system generates secure documents using the XML encryption. The XML signature is used in their work for signing the documents before sending them to the intended recipients.

Somorovsky et al (2010) proposed a streaming-based Web Services Security Gateway (WeSSeGa) in SOAP messages. They provided a comparison between WeSSeGa and Java XML Digital Signature API. From their experimented works, they showed that, the streaming-based approach provides more performance improvements with respect to memory consumption and the evaluation time. Chen-xi et al (2010), based on the conic over the ring $\mathbb{Z}_n$, proposed an improved ElGamal digital signature scheme, named CCC-i-ElGamal, which achieves more security than the original ElGamal schema. Juan & De-ting (2010) proposed an enhanced query-processing algorithm for encrypted XML data, using hash tables and chained lists to create the indexes for user keys. Their algorithm permits users to decrypt only the needed parts to the query result. Moreover, their algorithm disseminates an encrypted XML index with the encrypted XML data. Yue-sheng et al (2010) presented an overview about the XML signature and encryption technologies and described the steps of applying the XML signature and encryption.
Hao-yu et al (2011) proposed a new encryption model by combining symmetric key and public key encryption technologies to provide a shared symmetric key distribution algorithm. In their model, the symmetric key is used to encrypt the sensitive data, and the public key is used to encrypt the symmetric key in order to ensure the security in data transmission. In their model, the receiver has to use his private key to decrypt the symmetric key. This symmetric key can be used to decrypt the encrypted data. Similarly, the sender uses his private key to verify the identity of the sender and to maintain data integrity. Ladan (2011) classified the new levels of threats in his paper into service level and message level threats. In each class, the author described many threats and discussed about the existing mechanisms to handle them. Onashoga & Sodiya (2011) proposed a security system on an examination application, which transfers the examination results in a secure way. They have adopted the XML encryption and signatures to ensure the security goals, namely, confidentiality, integrity, authenticity, and non-reputability, while exchanging electronic results.

Chang & Hwang (2011) proposed a query-processing model for processing the encrypted XML documents, using the XQuery language effectively. Their model translates automatically the XQuery statements for the encrypted XML documents by applying DSL rules to encrypt the XML documents and Schemas, based on the original XML documents. Xiang & Wang (2011) proposed an asymmetrical encryption algorithm, called XRSA, based on XML. It is a combination between the asymmetrical encryption algorithm RSA and an XML encrypter device. Liu & Chen (2011) described a prototype of XML security for a certificate management and presented the design and implementation of the XML signature and encryption. Jensen & Meyer (2011) described the techniques of attackers to intrude to web services communication even in the presence of the XML signatures and described the interrelation between the XML signatures and encryption. Ammari et al
(2011) proposed a new model to act as an Intelligent XML tag classification model for the XML encryption. Their model proposed mainly to improve the security and efficiency of an XML messaging system. Their model uses on-the-fly mechanism for classifying XML messages, creating three layers, and applying fuzzy logic approach to determine which parts of the XML message needed to be secured depending on importance level attribute.

Seak & Siong (2011) demonstrated that, the applying of the XML encryption and decryption to any binary document would not change the integrity of it. Jager & Somorovsky (2011) proposed an attack model to enable an intruder to decrypt arbitrary data that are encrypted according to the XML encryption, based on a cryptographic weakness of the Cipher Block Chaining (CBC) mode. Fu & Wei (2011) proposed a sequential multi-signature scheme through a middleware technology to implement it in the electronic document system to sign and verify the HTML documents. Song & Cui (2011) proposed an electronic voting scheme of signature based on the ElGamal blind-signature algorithm. Their program has the ability to solve voters' fraud in thee-voting and to prevent multiple votes from the same voter. Pin-ai & Xiang (2011) analyzed XML security and described an approach on how to apply the XML security technology to the practical integrated circuit card system, to provide confidentiality, integrity, authenticity, and non-repudiation of the campus smart card system.

Li & You (2012) represented the log of typesetting on the web, as an XML document, and applied the XML signature technology to protect it. In their work, the .NET Framework is used for implementing the XML digital signature and validation of the log of typesetting on the web, using C# programming language. Mahfoud & Imine (2012) proposed an approach to handle query recursive XML views in a secure way using only the expressive power of the standard XPath. Geric & Vidacic (2012) presented the Information system architecture and the implementation of a web service.
They explained the difference between the XML digital signature and standard digital signature. Somorovsky & Schwenk (2012) analyzed the countermeasures against the new chosen-ciphertext attack on the XML encryption proposed in Jager & Somorovsky (2011), and they showed the reasons on why the countermeasures cannot handle this attack. Moreover, they have proposed two practical countermeasures against it. Wu et al (2012) proposed a technique for the analysis of the OpenXML based Office series encryption mechanism. Quevedo-torrero & Erickson (2012) presented a querying implementation of XML. Their framework exploited the Prolog based data structures leading to the handling of deductive and recursive queries.

Liu & Chen (2012a) proposed a method to transform XPath to XML data, using a hashing technique to provide effective integrity checking for decomposed XML data. They provided a signer to check the integrity without cooperation from other signers. Liu & Chen (2012b) proposed a number list based interval-labeling scheme for XML data encryption. When there is no space available for node insertion, the schema assigns a number for the sub-tree to be inserted, and performs a new labeling process for each node in the sub-tree. The main advantage of their model is its ability to handle updates effectively. Nithin & Bongale (2012) proposed a new public key cryptographic algorithm, named XML Batch Multi-Prime RSA (XBMRSA) to encrypt the XML documents. Their algorithm is based on Multi-Prime RSA technique. The main idea is using multiple of prime numbers to compute the modulus for both the public and private keys (N), instead of two prime numbers as in the standard RSA. Their techniques needs less computation time for the encryption of the XML documents, and it is more efficient public key cryptographic algorithm than the Standard RSA algorithm. Algarin et al (2012) presented a new UML class diagram, called an XML Schema Class Diagram (XSCD) to transition an XML schema into an UML schema.
Moreover, they defined a new UML XML Role Slice Diagram (XRSD) that allows permissions to be defined against XML schema elements in the XSCD. Finally, they transited these XSCDs into a corresponding security policy to generate automatically an XACML policy for enforcement of the XML schema at the instance level. Li-yan & Huan (2012) proposed a design and realization of electronic commerce platform based on the XML signature.

Cao et al (2013) proposed an efficient evaluation of tree pattern queries on the encrypted XML documents. They embedded each XML document in a hierarchy and created a vector, which encoded the information about each XML document. Moreover, they created a vector for tree pattern queries and matched between the two encrypted vectors.

2.2 ACCESS CONTROL TO XML DOCUMENTS

Tan et al (2001) proposed an algorithm, called XML sEcurity eNforcement Architecture (XENA). XENA is an access control system for XML documents stored in relational database. The XENA maps XML documents in relational tables and uses authorization rules to designate the information that must be protected. Moreover, XENA verifies the retrieved information against the authorization rules to filter the protected information. XENA conducts to the access control rules from outside the relational engine, which leads to a performance overhead.

Murata et al (2003) proposed a static analysis model for XML access control. However, their technique increases the time complexity, while checking the security of queries when static analysis fails. Fan et al (2004) proposed a complex and an expensive technique to express access control policies using XPath queries. Wu et al (2005) proposed a new Authorization Policy Sheet (APS) tool to describe the authorization rules for the XML documents. Moreover, the Document Type Definition (DTD) is used to
translate the rules described in an APS into a standard XML document. Kuper et al (2005) proposed a generalization of XML security views. Their model specifies the policies over DTDs with XPath expressions. Mohan et al (2005) proposed a technique for security views. Their technique is restricted on hiding node values.

Fan et al (2006a) proposed an XML security framework to protect sensitive data from direct access or indirect inference through queries by unauthorized users. Their framework supports fine-grained access policies according to the structure and values of the protected XML data. Moreover, their framework assists the security administrators to derive views specification automatically. Finally, their framework provides a view schema to different user roles, which makes it possible to guide users when they are running queries on the system. Fan et al (2006b) proposed a Secure MOdular Query Engine (SMOQE) approach to rewrite, evaluate, and optimize queries on XML views. Lv & Yan (2006) proposed a framework web, called Web Application Level Security Gateway (WALSG) to provide web security. Their framework is based on the two proposed languages, namely, Access Control Policy Description Language (ACPDL) and Security Policy Description Language (SPDL). Their framework is capable for securing the existing web sites and creating new web sites in a secure way.

Ganesan & Mohamed Jamal (2006) proposed an integration approach between the bitmap based access control approach and cryptography. Their integration allows to the generation of the ePath level access control, using two dimensional access control matrix, instead of using three dimensional security cubes. Therefore, their approach saves the redundant storage space needed at the server for storing the restricted views of the XML documents. Sanchez et al (2006) proposed a simple approach to the XACML policies specification, based on the use of high-level templates and
Infopath application. An XACML policy is generated automatically, using an XML transformation as the templates that are built from XML schemas. However, they neither provided any details about the methodology nor the practical implementation of their approach. Anderson (2006) presented a comparison between the policy language, namely, Enterprise Privacy Authorization Language (EPAL) and XACML. The two languages have been developed for expressing the privacy policies. The conclusion of the comparison ensures that, the functionality of XACML is more powerful than EPAL.

Fu & Ye (2007) proposed an access control model, based on the users' and roles' attributes. Their model uses users' and roles' attributes as part of the access control policy. Moreover, they proposed an XACML-based policy language, namely, A-XACML based on XACML. Ro’der et al (2007) proposed a model for access control to the XML documents, based on the current document content and history information that recorded the operations performed on that document. Chang et al (2007) solved the limit of Mohan et al (2005) by considering constraints based on structural relationships between elements.

Kim et al (2008) proposed an algorithm to store the XML data and XML access control rules in a relational database. Their algorithm allows users to exploit the features of relational database systems, to check the security of the XML documents, and run queries on them. Lang et al (2008) proposed an XACML policy generating method, based on a user-oriented ABAC policy view. Moreover, they proposed a new ABAC concept model, called Access Control Cube (ACCube) and presented a policy description template composed of primary policy description elements of XACML. Khurat & Abendroth (2008) proposed a mechanism where the Policy
Decision Point (PDP) evaluates the request only once when multiple resources are requested, hence, the processing time is reduced.

Scaglissi et al (2008) presented a complete description of XACML and the implementations of the basic XACML components. In addition to the description, they listed the XACML benefits, drawbacks, and open issues to improve XACML. Sasaki et al (2008) proposed a fine-grained access control model, based on logic programming in hybrid relational XML database. Their model is capable for restricting the access to the relational and XML data. For the relational data, the restriction is applied at the cell-level. However, it is applied at the node-level for the XML data in a cell. Li et al (2008) proposed an approach to find the defects of the XACML implementations. Their approach determines the defects by testing the behaviors of different XACML implementations for the same XACML policies and requests, and observing whether the different XACML implementations produce different responses or not.

Peng et al (2008) proposed an approach to match dynamically the semantics of the users and information in the XML documents to control the access to them. Rahaman et al (2008) proposed a distributed and fine-grained access control mechanism, based on the encryption for the XML documents. The proposed technique achieves the confidentiality, authenticity, and integrity. Karusseit et al (2008) presented a comparison between XACML and WS-Policy with the constraints and graph-based approach for modeling access control. Duong & Zhang (2008) proposed a fine-grained access control model, namely, SecureX to query and update XML data securely. Their model allows users to read and write, and it is capable for defining rules explicitly for the users to authorize the access to XML data. Li & Hong (2008) proposed an extension to the authorization term to be AUTH (Auth, Autho). Their extension presents the time information in the authorization
rules. Moreover, they proposed a bitmap indexing technique with time information and proposed an authorization-processing algorithm, to retrieve the correct authorization decision.

Bouganim et al (2008) proposed a streaming evaluator of access control rules, to provide a powerful fragment of the XPath language. They designed a streaming index structure allowing overtaking the unauthorized parts of the input document. They proposed a technique for the management of pending predicates compatible with a streaming delivery of the authorized parts of the document. They proposed a secure mechanism to refresh the SOE access control rules from a potentially malicious server. Moreover, they proposed a combination of hashing and encryption techniques to ensure the integrity of the document. Mazzoleni et al (2008) proposed an integrating algorithm, to integrate two well-formed XACML policies. Mohy & El-Sharkawi (2008) proposed an approach, namely, Disclosure Prevention Algorithm (DPA). Their approach combines the role based access control models with the power of inference engine, to prevent unauthorized users from inferring critical information.

Qu et al (2009) discussed about the access control and security specifications, defined on DTD with production expression, and explained how to construct security XML views. Laborde et al (2009) extended the XACML authorization web service by adding a core element to it that implemented both XACML and additional modules, for providing new security features for securing information. Their extension is applied to dynamic web sites access control management. Hsieh et al (2009) proposed an extension to the XACML policy language by adding a <ResourceContent> within a <Resource>. Aburahma & Stumptner (2009) proposed a Spatial Role-Based Access Control (SRBAC) model, using XACML. Their model is an extension of RBAC to combine location information in access control decisions. Ardagna et al (2009) proposed a privacy-aware access control
system. Their system is implemented in PrimeLife project and is compatible with XACML. Koromilas et al (2009) proposed a re-annotation algorithm to control the access to the XML documents stored in relational and native XML databases. While the update operation, their algorithm runs the XPath query, that retrieves the XML nodes to be re-annotated.

Lischka et al (2009) proposed the concept of deductive policies in XACML that permit to specify policies on the Software as a Service (SaaS) service. Their deductive policies help to deduce the decision. Mondal & Sural (2009) proposed an XML based policy specification framework, namely, Enhanced Spatio-Temporal Role Based Access Control (ESTARBAC) for spatio-temporal RBAC model. Their framework describes the spatio-temporal extent. This extent provides a variety of spatio-temporal access control policies, such as role hierarchy, separation of duty, and cardinality. Ferrini & Bertino (2009) proposed a new framework by integrating XACML and Web Ontology Language (OWL) frameworks. Therefore, it provides the features of both OWL ontologies and XACML policies, for supporting RBAC. OWL handles the role hierarchy and constraints and XACML handles the authorization policies.

Dai et al (2010) proposed the architecture of Usage Control (UCON) model in web services and explained access control models and XACML. Jing et al (2010) provided a description for a general security policy, namely, XML-Based General Policy Description (XBGPD). They defined the behavior of the system’s entities. Moreover, they discussed the logic relationship in policy’s element and showed XML-based description rules about the policy. Landberg et al (2010) proposed a novel privacy-aware access control model for XML. Their model uses the query-time access control for combined access to nodes. Moreover, they proposed a security level composition to group nodes in the XML data. Abassi et al (2010) made a comparison between XACML policies and
annotated schemas and identified the significant fragment of XACML. Moreover, they proposed translation algorithms from XACML policies into annotated schemas.

Jinsong et al (2010) proposed an access control model, based on RBAC to handle the access of the users to the resources and authorize the users' rights. Their model is implemented in the XML documents for complex business process using J2EE. Hsieh et al (2010) proposed a combination of XACML, XML encryption, and XML digital signature, to provide a secure, embedded, and fine-grained access control policy. Their combination allows for embedding the content to be protected with the access control policy statements in the same XACML document. In their model, the content is protected by the XML encryption for confidentiality and by the XML digital signature for authentication and integrity. Arunkumar & Rajarajan (2010) proposed an architecture to access control the data privacy through mobile devices, using XACML policy. Alm & Illig (2010) proposed a translation approach from ORKA Policy Language (OPL) into XACML. In their approach, the administration of high-level access control concepts and the authorization constraints are combined together with XACML. The aim of their approach is to solve the problem of the XACML policies administration.

Rota et al (2010) proposed an approach to combine between the advantages of the XACML policy language and OWL. Moreover, they proposed a practical privacy filtering application to filter out the required information from the XML documents with respect to a set of XACML semantic privacy policies. Bekara et al (2010) proposed a privacy aware XPACML policy language model to combine the advantages of XACML and Platform for Privacy Preferences Project (P3P). Their model allows the users and service providers to define their privacy and policies in XACML format. Chou & Huang (2010) proposed an extended XACML model, namely,
EXACML to provide more secure information access to web services. Their model is an extension of XACML and based on the concepts of information flow control to avoid the leakage of information. Helil & Rahman (2010a) tried to solve the XACML complexity by proposing an extended XACML profile for RBAC by using an OWL base approach. Helil & Rahman (2010b) proposed an extension to the XACML profile for RBAC to provide constraints, such as static and dynamic separation of duty and role cardinality. Moreover, they presented an analysis for XACML profile for RBAC model. Ardagna et al (2010) proposed an extension to the XACML rules with two child elements <CredentialRequirements> and <ProvisionalActions>. The former describes the credentials that the requester needs to own and the conditions that must be satisfied. The latter describes the actions that the requester has to perform.

Ran & Guo (2011) integrated a traditional XACML and scalable SOAP message-level security strategy to construct a security XACML access control model based on SOAP encapsulate (S-XACML). In their model, the XACML message is packaged into a SOAP message. Patel & Atay (2011) proposed an access control model, namely, XML to Relational Authorization Rule (X2RAC) for relational storage of the XML documents. Their model permits the security administrators to specify all the authorization information in XML, and store them into relations. Mourad et al (2011) proposed an approach to create XACML policies from XACML profiles. Their approach is based on the preparation of an abstract language on top of an XACML profile specification, which is translated into an XACML policy, using their compiler. Xu et al (2011) proposed an XACML-ARBAC profile to specify Administrative Role Based Access Control (ARBAC) policies. Moreover, they extended the Sun's XACML architecture by developing an Administrative Policy Enforcement Point (APEP) and a Lock Manger to provide the security features of a policy management.
Stepien et al (2011) proposed the non-technical XACML presentation notation, to ease the complexity of policy rules. Farooqi & North (2011) used the trust-based access to XML databases. In trust-based access control, threats are detected, user privileges are calculated dynamically depending on the users trust value, and the access decision is depended on matching the node trust value and the user trust value. Ma et al (2011) proposed an architecture for sharing information cross social networks, based on XACML securely and effectively. An & Park (2011) proposed an efficient access control labeling scheme. Their schema is used for processing XML queries under dynamic XML data streams securely. Their schema is based on the dynamic roles generated using prime number and group-based prime number labeling schemas where labels are encoded with ancestor-descendant and sibling relationships between nodes. Luo et al (2011) proposed an efficient technique, named QFilter, based on Non-Deterministic Finite Automata (NFA). Their technique translates the user's insecure queries into secure ones, without violating the access control rules.

Li & Fan (2012) proposed a combination of the RBAC model and a specific credit management system, to instantiate the model. Moreover, they used UML to explain the model, and XACML to explain the access permission between the users and objects. Chae et al (2012) proposed role attributes for access control in XML databases in which attributes are used to specify the characteristics of a role. Their method extends the XACML RBAC profile to support role attributes by proposing new entities, namely, Role Attribute Point (RAP) and Extended-PIP (E-PIP). RAP is used to manage the proposed role attributes. Extended-PIP (E-PIP) is an extension of the Policy Information Point (PIP) of XACML. It is connected to RAP to deal with role attributes. Farooqi & North (2012) used the Xlog file as a dynamic and temporary log file for XML databases. They used the Xlog files to evaluate user' behavior by recording user' transactions and errors.
Yang & Liu (2012) proposed an action type of access control model. Their model defines the type of behavior to solve the problem of the access model. Xin-fang & Xiao-hua (2012) proposed an extension to the attribute-based access control model using hidden credentials technology. Their extension provides the cross protection of sensitive attributes and strategies to perform attribute-based access control. Chebotko et al (2012) classified the access control models to the XML documents into XPath-based and DTD-based model. They defined each class and listed the advantages and disadvantages for each class. Moreover, they proposed the first DTD-based access control model that exploited the graph matching to analyze if an input query is fully acceptable, fully rejectable, or partially acceptable. Therefore, the fully acceptable and rejectable queries are not needed for checking. However, the partially acceptable queries will be rewritten into an equivalent recursive query or into a non-recursive query.

Stepien et al (2012) proposed an algorithm that used a recursive process of subsumption carried out on the original set of policies for reducing the size of access control policies in XACML. The reduced policy sets decrease the risk of conflicts and improve PDP performance. Thi & Dang (2012) proposed an access control model, namely, eXtended XACML for Spatial Temporal Role based access control model with OWL (X-STROWL). They combined XACML and OWL ontology for providing the NIST standard RBAC model with new data types and functions. Bravo et al (2012) presented techniques to solve inconsistent full and partial policies. They proposed approximate algorithms and showed that, they got reasonable results in practice. Moreover, they evaluated the algorithms and showed that, the consistency and insert–update-delete repair algorithms are running fast and effectively. Ulltveit-moe & Oleshchuk (2012) proposed a decision cache for fine-grained XACML authorization and anonymization of elements and
attributes in the XML documents. Their model permits for management of authorization and anonymization policies for the XML documents.

Thimma et al (2013) proposed a hybrid XML access control mechanism, namely, HyXAC. The proposed technique uses QFilter approach to process queries. It defines a sub-view for each access control rule. Moreover, it allocates dynamically the available resources to materialize and cache sub-views to improve query performance. Laborde et al (2013) studied the using of XACML for Authorization as a Service in the cloud security community. Bertolino et al (2013) proposed a set of mutation operators to check the faults of the XACML 2.0 policies. Moreover, they proposed a tool, called XACMUT (XACmlMUTation), for creating mutants. Lin et al (2013) proposed a policy similarity measure approach, to determine similar policies.

2.3 PROPOSED WORK

In spite of all these works, the temporal attributes were considered as any other attributes. However, it is necessary to consider the temporal attributes as special attributes, due to their importance in many applications. In addition to the temporal attributes, the security features of XML databases are violated or tampered with by malicious users. Therefore, secure and temporal oriented algorithms are proposed in this research work, to enhance the security and applications' requirements.

Moreover, none of these works focused on how to overcome the complex structure of XACML, which is useful to express and interchange access control policies and requests/responses effectively. Hence, this research proposes new algorithms for mapping XACML policies and rules into relational rules, and storing them in the form of rules in temporal relations, to ease the access control to the XML documents. This proposed work relieves the users from the effort of learning and understanding the
XACML policies and rules; hence, it reduces the users' time and effort. It controls the access to the XML documents stored in either native or relational databases, using the XACML policies. Finally, it applies the constraints of rules, and obligations and provides the response to an access request effectively.

The second contribution of this research work is proposing a new time-stamp based algorithm, for decrypting the encrypted XML documents partially. This proposed algorithm utilizes a new temporal model for XML representation, based on a combination of valid time and transaction time, through the tuple time-stamping. Moreover, one attribute of the key components of the encrypted elements is selected to distinguish among them. The time-stamp and the selected attribute guide the receiver to decrypt the required parts of the encrypted XML documents, instead of decrypting all the parts of them. Moreover, they can be used to retrieve the parts that must be decrypted, to contribute to the query results on the encrypted XML documents. The proposed algorithm not only reduces the decryption time, but also protects the security of the data.

2.4 SUMMARY

This chapter presents a literature review of XML security (signature and encryption) and access control to the XML documents. This survey explains that, the temporal attributes were not considered as special attributes, the security features of XML databases are violated or tampered with by malicious users, and there is no proposal about how to overcome the complex structure of XACML, which is useful to express and interchange access control policies and requests/responses effectively.
CHAPTER 3

XML SIGNATURE AND ENCRYPTION

There are two types of security ranges in Internet security, namely, Point-to-Point, and End-to-End security. The former ensures the security between two adjacent nodes of the network. The latter ensures the security from the initial sender until the final recipient, which is more secure for web services. The Secure Socket Layer (SSL) provides a Point-to-Point security. SSL is not suitable for the transmission modes of web services, such as the Transmission Control Protocol (TCP), the File Transfer Protocol (FTP), and messages formation. SSL can execute the encryption of complete information, but it cannot execute the encryption of partial information. SSL guarantees point-to-point security, but it does not guarantee end-to-end security. It provides confidentiality, authentication, and integrity.

On the other hand, XML security is a representative of End-to-End security. XML security is flexible for the application, especially, of mobile web services that demand more flexible, customizable, and better-optimized security schemes. XML security provides the following services (Sun & Li 2008, Knap & Ml’ynkov´a 2009a, Yue-sheng et al 2010, and Hao-yu et al 2011):

1. Confidentiality: Ensuring that only the intended receiver will read the transmitted document, and others cannot access or copy the data.

2. Integrity: No change in the transmitted document from the source to the final destination.
3. Authenticity: Determining that, a user has a genuine identity.

4. Non-repudiation: The sender cannot disclaim his responsibility for sending the document.

Therefore, the concern of XML security has been raised to a significant level, focusing on methods and approaches, to secure the XML messages exchanged.

This chapter presents two XML security technologies, namely, XML signature and encryption. It presents an overview of how they integrate with XML in such a way, as to maintain the advantages and capabilities of XML, while adding the necessary security capabilities.

3.1XML SIGNATURE

The XML signature, called XMLDsig, XML-DSig, or XML-Sig, defines the XML syntax for digital signatures, and is defined by W3C and the Internet Engineering Task Force (IETF) in Bartel et al (2013). The XML signature creates a highly extensible signature syntax, which is integrated tightly with the existing XML technologies. The XML signature is a digital signature obtained by applying a digital signature operation to XML resources. Moreover, the XML signature can be applied to any digital data or binary data, such as a JPEG-file. Its functionality is similar to that of the PKCS\#7 (Kaliski 1998).

The existing technologies allow us to sign only a whole XML document. However, the XML signature provides a means to sign the entire document, parts of a document, or multiple signatures written in the same document. This functionality is very important in a distributed multi party environment, where the necessity to sign only a portion of a document arises, whenever changes and additions to the document are required. The XML
signature has been used to solve security problems, such as falsification, spoofing, and repudiation, by ensuring confidentiality, integrity, authenticity, and non-repudiation.

3.1.1 Syntax

The structure of the XML signature document is depicted in Figure 3.1, as defined by Bartel et al (2013):

```xml
<Signature Id?>
  <SignedInfo Id?>
    <CanonicalizationMethod Algorithm/>
    <SignatureMethod Algorithm/>
    (<Reference Id? URI? Type?>
     (<!--Transforms>
      (<Transform Algorithm />)+
     </Transforms>)?
    <DigestMethod Algorithm />
    <DigestValue>
    </Reference>)+
  </SignedInfo>
  <SignatureValue>
    (<KeyInfo>
     <choice>
      <KeyName>*
      <KeyValue>*
     </choice>
    </KeyInfo>)?
    (<Object Id?>)*
  </SignatureValue>
</Signature>
```

Figure 3.1 XML Signature's Structure
In Figure 3.1, the <Signature> is the root element of an XML signature. It is composed of one <SignedInfo>, one <SignatureValue>, zero or one <KeyInfo>, and zero or more <Object> elements. Moreover, it has an optional Id attribute, which allows the <Signature> to be referenced by other signatures or objects (Bartel et al 2013).

3.1.1.1 Signed Information

The required <SignedInfo> includes the data objects, which have been signed. The <SignedInfo> contains one <CanonicalizationMethod>, one <SignatureMethod>, and one or more <Reference> elements. The <SignedInfo> may include an optional Id attribute, which permits it to be referenced by other signatures or objects. The <SignedInfo> does not include explicit signature or digest properties, such as date/time of the signing process.

The required <CanonicalizationMethod> determines the canonicalization algorithm, which is used to canonicalize the <SignedInfo> before the proceeding of the signature calculations. It has a required attribute, called an Algorithm, which specifies the URI (Uniform Resource Identifier) of the used algorithm in the W3C specification.

The required <SignatureMethod> designates the algorithm used for creating the signature and performing the validation of the canonicalized <SignedInfo>. It includes an Algorithm attribute, which specifies the URI of the cryptographic algorithm.

The required <Reference> includes the digest method, and the digest value calculated over the identified data objects. The structure of the <Reference> contains optionally, an Id attribute, a URI attribute, and a Type attribute. Moreover, it contains zero or one <Transforms>, one
<DigestMethod>, and one <DigestValue>. The optional Id attribute allows the <Reference> to be referenced by other signatures or objects. The optional URI attribute identifies a data object that will be signed, using a URI-Reference. For example, URI = "http://Company.com/emp.xml" means that, the emp.xml document will be signed. The URI attribute and <Transforms> describe the retrieving and preparation of the data objects, which will be digested (i.e., the input to the digest method). The optional Type attribute provides information about the data objects, obtained by the URI attribute, to ease the processing of the referenced data.

The optional <Transforms> contains one or more ordered <Transform> elements, which are applied to the original data objects before they are digested. These ordered elements describe the preparation of the original data objects before digesting. The transform operations include canonicalization, encoding/decoding, XSLT, XPath, XML schema validation, or XInclude, which are applied to the original data. The <Transform> includes a required attribute, called an Algorithm, which indicates the operation that will be applied to the original data objects. If the <Transforms> is omitted, the data objects are digested directly without any transformation. The output of each <Transform> is the input to the next one. The input to the first <Transform> is the original data objects obtained by the URI attribute of the <Reference>. The output from the last <Transform> is the input for the <DigestMethod>. When the transform operations are carried out on the data objects, the signing is not done on the original data objects but on the resulting transformed data objects.

The required <DigestMethod> specifies the digest algorithm that will be applied to the transformed data objects, to get the digest value that will be signed. The required <DigestValue> includes the encoded digest value, using base64 that will be signed.
3.1.1.2 Signature Value

The required <SignatureValue> includes the encoded value, using base 64 of the digital signature.

3.1.1.3 Key Information

The optional <KeyInfo> specifies the needed key to the recipients to validate the signature. It includes keys, names, or certificates. The <KeyInfo> contains multiple <KeyName> or multiple <KeyValue> elements. Only one of them appears in the <KeyInfo>. If the recipient knows the key, the <KeyInfo> can be omitted. The <KeyInfo> is optional for the following reasons:

1. The signer may not want to disclose the key.
2. The key information may exist implicitly in the application's context.

If the signer needs to attach the key information to the signature, a <Reference> can get and contain the <KeyInfo> as part of the signature. The <Reference> is used, because the <KeyInfo> is outside of the <SignedInfo>. The <KeyName> includes a string value, and a whitespace is allowed. It is used to identify the key for the recipient. The <KeyValue> includes a public key value of the signer that is exploited in validating the signature. The <KeyValue> may include externally public key values as Parsed Character Data (PCDATA), or element types from an external namespace.

3.1.1.4 Object

Since the <SignedInfo> does not include explicit signature or digest properties, if an application needs to associate properties with the signature or digest, it may include such information in a <SignatureProperties> within an
<Object>. The optional <Object> includes any data (signature properties) within the signature. The <Object> includes an optional Id attribute used to refer to it by the URI attribute of the <Reference> to be signed in the enveloping signature. Moreover, the URI attribute of a <Reference> can refer to the Id attribute of the <SignatureProperties> in the <Object> to be signed in the enveloping signature.

3.1.2 XML Signature Types

There are three types of XML signature, namely, enveloping, enveloped, or detached signature (Qiao 2007, Bartel et al 2013). In the enveloping signature, the <Signature> contains the signed data; the signed data is included within an <Object> or a <SignatureProperties>. The URI attribute of the <Reference> identifies the data by referring to the Id attribute of the <Object> or of the <SignatureProperties>. An example of the enveloping signature is depicted in Figure 3.2:

```
<SignedInfo>
  <CanonicalizationMethod
    Algorithm = "http://www.w3.org/2006/12/xml-c14n11"/>
  <SignatureMethod
    Algorithm = "http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
  <Reference
    URI = "#obj">
    <DigestMethod
      Algorithm = "http://www.w3.org/2000/09/xmldsig#sha1"/>
  </Reference>
</SignedInfo>
</Signature>
```

Figure 3.2 (Continued)
Figure 3.2 Enveloping XML Signature

In Figure 3.2, the data referenced in the `<Reference>` by URI = "#obj" is the value of the Id attribute of the `<Object>` to get the data residing in it that have been signed.

In the enveloped signature, the signed data is an XML document, containing the `<Signature>` as its child element. The Signature calculation must be performed on the XML document except the content of the
<Signature>; thus the content of the <Signature> is excluded from the calculations of the data digest and signature value, by using the enveloped-signature transform whose identifier is "http://www.w3.org/2000/09/xmldsig#enveloped-signature". An example of the enveloped signature is depicted in Figure 3.3:

```xml
<Message>
  <To> OMAR </To>
  <Body> Take Care </Body>
  <From> Zizo </From>
  <Signature xmlns = "http://www.w3.org/2000/09/xmldsig#" >
    <SignedInfo>
      <CanonicalizationMethod Algorithm="http://www.w3.org/2006/12/xml-c14n11"/>
      <SignatureMethod Algorithm = "http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
      <Reference URI = ""/>
      <Transforms>
        <Transform Algorithm = "http://www.w3.org/2000/09/xmldsig#enveloped-signature"/>
      </Transforms>
      <DigestMethod Algorithm = "http://www.w3.org/2000/09/xmldsig#sha1"/>
      <DigestValue>...</DigestValue>
    </SignedInfo>
    <SignatureValue>
gjggjmdorldmfl
    </SignatureValue>
    <KeyInfo>... </KeyInfo>
  </Signature>
</Message>
```

**Figure 3.3 Enveloped XML Signature**
In Figure 3.3, the data referenced in the <Reference> by URI = "" is the entire <Message> (all the document) including the <Signature> itself. However, the value "http://www.w3.org/2000/09/xmldsig#enveloped-signature" of the Algorithm attribute of the <Transform> ensures that, the <Signature> and its content are excluded from the signature processing.

In the detached signature, the signature is carried out on external XML data outside the <Signature> or on local data objects that reside in the same XML document. The external data can be identified by the URI attribute of the <Reference>. For the local data objects, the <Signature> and local data objects will be two sibling elements in the same document. The following figures are two examples of the detached signature for external and local data:

```xml
<Signature  xmlns = "http://www.w3.org/2000/09/xmldsig#">
  <SignedInfo>
    <CanonicalizationMethod  Algorithm = "http://www.w3.org/2006/12/xml-c14n11"/>
    <SignatureMethod  Algorithm = "http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
    <Reference  URI = "http://www.w3.org/TR/2000/REC-xhtml1-2000112"/>
    <DigestMethod  Algorithm = "http://www.w3.org/2000/09/xmldsig#sha1"/>
    <DigestValue> flfjfljfjcyBpeyBub3QgYSBzaWduY </DigestValue>
  </Reference>
</SignedInfo>
<SignatureValue> KHGBNMLO </SignatureValue>
</Signature>
```

Figure 3.4 Detached XML Signature for External Data
In Figure 3.4, the data referenced in the <Reference> by URI = "http://www.w3.org/TR/2000/REC-xhtml1-2000112" is the external data that have been signed.

```xml
<SignedInfo>
  <CanonicalizationMethod Algorithm = "http://www.w3.org/2006/12/xml-c14n11"/>
  <SignatureMethod Algorithm = "http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
  <Reference URI = "#localdata">
    <DigestMethod Algorithm = "http://www.w3.org/2000/09/xmldsig#sha1"/>
    <DigestValue>dfjkdhfkdfhjkdfhjkd</DigestValue>
  </Reference>
</SignedInfo>

<SignatureValue>jdhasjdjasdhjaskd</SignatureValue>

<LocalData Id = "localdata">
  <title>XML security</title>
  <author>Prof. A. Kannan</author>
</LocalData>
```

**Figure 3.5 Detached XML Signature for Local Data**

In Figure 3.5, the <Signature> and the <LocalData>, which is the local signed data, are sibling elements in the XML document.

### 3.1.3 Canonicalization

The contents of digital signatures must be identical on signature application and verification. Otherwise, the digital signatures will be invalid.
In the XML digital signature, slight changes to the structure of the signed XML document are acceptable, as long as the document's contents keep identical to an XML parser. For example, XML comments or whitespaces between an element's attributes are not considerable to an XML parser. Therefore, these modifications will not affect the validation of the signature. This flexibility of XML signature is due to the concept of canonicalization.

The canonicalization is a set of transformations that are applied to the XML contents before digesting. The purpose of canonicalization is to hide the irrelevant modifications to the XML document, such as removing in significant whitespaces, sorting attributes, sorting namespace declarations of each element, and adding fixed attributes according to the DTD file that will not affect the verification of the signature (Jensen et al 2009, Knap & Mi’ynkov’a 2009b).

In other words, XML canonicalization is significant, because two XML documents may be logically equivalent, but differ in their physical representations. The differences in their physical representations may be due to syntactic changes, permitted by XML and namespaces in XML. These differences in their physical representations will lead to different digest values of such XML documents, although they have the same content. Therefore, the canonicalization methods define a physical representation, namely, the canonical form, and the logically equivalent documents will be transformed to the canonical form, to have the same physical representation (Nordbotten 2009). For example, let there be two XML documents, namely, Doc1 and Doc2 containing an element B, which has two attributes, A1 and A2. Doc1 has a comment element, and defines attributes A1 and A2 respectively. However, Doc2 defines the attributes in the reverse order; i.e., attribute A2 and then attribute A1. The physical form of these two XML documents is different, but the logical form is the same. Hence, the canonicalization process
will transform these two logically equivalent XML documents to the same physical form. Therefore, the digesting process of the two XML documents will return the same values (Knap & Ml’ynkov’a 2009b).

There are three W3C Recommendations for the normalization process, namely, XML Canonicalization 1.0 (Can1.0) (Boyer 2001), XML Canonicalization 1.1 (Can1.1) (Boyer & Marcy 2008), and Exclusive XML Canonicalization 1.0 (ECan) (Boyer et al 2002). ECan is based either on Can1.0 (denoted as ECan1.0) or on Can1.1 (denoted as ECan1.1). Can1.0 is the original W3C Recommendation, which defines the whole normalization process. Can1.1 is an enhanced version of Can1.0, which addresses the issues with the processing of the attributes, namely, id and base from the XML namespace. Can1.1 and ECan1.1 differ in the processing of the declared namespaces. Can1.1 includes the result of the normalization of an XML element that will be signed by all the namespace declarations in its parent element, although its parent element will not be signed. However, ECan1.1 contains only those namespace declarations, which are required for parsing the signed contents, and to have only their namespace declarations become embedded in the signed document. Hence, In ECan1.1, the namespace declarations of the enclosing XML element are not involved in the result of the normalization process, unless they are used in the signed XML element.

3.1.4 Processing and Implementations

In this work, XML documents are signed and verified by applying the following steps (Liu & Chen 2011):

1. Specify the location of each XML fragment that will be signed, and assign it to a URI attribute of a <Reference>. 
2. Apply the ordered transform operations to each XML fragment that will be signed (optional).

3. Assign each used transform operation to an Algorithm attribute of a <Transform>.

4. Add the ordered list of the <Transform> in a <Transforms>.

5. Apply a hashing algorithm to each transformed XML fragment and get a digest value for each fragment.

6. Assign the used hashing algorithm to an Algorithm attribute of a <DigestMethod>.

7. Add each calculated digest value in a <DigestValue>.

8. Add the <Transforms>, <DigestMethod>, and <DigestValue> of each XML fragment to its <Reference>.

9. Add the <Reference> of each XML fragment to a <SignedInfo>. Hence, the <SignedInfo> contains all the digested XML fragments.

10. Assign a canonicalization process to an Algorithm attribute of a <CanonicalizationMethod>.

11. Assign a signature algorithm to an Algorithm attribute of a <SignatureMethod>.

12. Add the <SignatureMethod> and <CanonicalizationMethod> to the <SignedInfo>.

13. Apply the selected canonicalization process to the <SignedInfo>. 
14. Apply the same hashing algorithm to the \(<\text{SignedInfo}>\) and get a digest value for it. Since the \(<\text{SignedInfo}>\) contains all the digested XML fragments, they are implicitly signed as well, when signing the \(<\text{SignedInfo}>\) using a digital signature algorithm, such as the RSA, DSA, or Elliptic Curve.

15. Apply the selected signature algorithm, using the signer's private key to the \(<\text{SignedInfo}>\) and get the signature value for the \(<\text{SignedInfo}>\).

16. Add the signature value in a \(<\text{SignatureValue}>\).

17. Add a \(<\text{KeyInfo}>\) to specify the signer's public key, which will be used to validate the signature (optional).

18. Add the \(<\text{SignedInfo}>\), \(<\text{SignatureValue}>\), and \(<\text{KeyInfo}>\) to a \(<\text{Signature}>\), which is the final resulting signature.

To verify a signature, we perform the following steps:

1. Apply the ordered transform operations specified in the \(<\text{Transforms}>\), which is within the \(<\text{Reference}>\) to each original XML fragment.

2. Apply the hashing algorithm specified in the \(<\text{DigestMethod}>\), which is within the \(<\text{Reference}>\), to each transformed XML fragment to be sure it is not changed, and get a digest value for each fragment.

3. For each transformed XML fragment, compare the calculated digest value with the one in the \(<\text{DigestValue}>\), which is within the \(<\text{Reference}>\). If they do not match, the signature verification fails. Otherwise, go to the next step.
4. Apply the canonicalization process specified in the <CanonicalizationMethod> to the <SignedInfo>.

5. Apply the hashing algorithm specified in the <DigestMethod> to the <SignedInfo>, to be sure it is not changed.

6. Retrieve the digest value from the signature value, which is included in the <SignatureValue> using the signer's public key.

7. Compare the retrieved digest value with the calculated digest value. If they do not match, the signature verification fails. Otherwise, the signature verification succeeds.

The abstract algorithms, such as MD5 or SHA-1, are used to calculate the hash value for a given input. In this work, the signature process is applied to the digest value, since the digest value is less than the original data. Therefore, applying the signature process has reduced the signing time and the space of the memory required for the data. Hence, the transmission of the data is effective. The digest value used in this work represents the fingerprint of the original data. If any slight modifications occurred to the original data, the digest value will have a huge change, because of the avalanche effect property of the abstract algorithm. After the digest value has been signed, the XML documents cannot be changed. Hence, the integrity of the documents can be fulfilled (Yue-sheng et al 2009).

Many library function tools assist researchers to implement the XML signature. They are (Haron et al 2010):

1. Java XML Digital Signature API: Sun provides a Java development kit for the developer to apply W3C recommendations for XML security technology specification,
by translating XML security to Java Specification Request 105 (JSR 105), which is Java APIs. The JSR 105 is a collection of technical specifications developed in Java and designed to conform to the XML signature specifications.

2. Apache XML Security Library: Apache XML security (http://santuario.apache.org) is an open source initiative for XML security launched under Apache Software license. It is developed in C++ and Java and provides the W3C specifications for the XML signature.

3. XML Security Library: The XML Security Library known as the XMLSec library (Sanin et al 2013) is an open source initiative to enable the integration of XML security inside a third party solution. It is developed in C++, launched under MIT license, and provides the W3C specifications for the XML signature in both Windows and Linux operating systems.

4. XML Digital Signature Tool Application: XML Digital Signature Tool (Mazumdar 2007) is a Firefox extension for applying the digital signature to XML documents. The XML interface is built on top of the Apache XML Security library. It is developed in C++ and Java Script languages and supports the W3C Specification for the XML signature.

5. XML Security Tool Application: The XML security tool plug-in (Schadow 2005) for Eclipse permits the users to utilize the XML security features. This application is launched under Eclipse Public License. This tool aims to ease the exploiting of the XML technology. The XML interface is built on top of the Apache XML Security library using Java.

7. JVNRS XML Signature: SIG rdf (RDF with XML signature) (Terada et al 2006) is an application to promote the use of the XML signature within a security information exchange.

3.1.5 Comparison with Traditional Digital Signature

The limitations of the traditional digital signature are as follows (Chen-xi et al 2010, Yue-sheng et al 2010):

1. The size of the signature is very large, and the unit is only a document.

2. The traditional digital signatures do not provide electronic documents, multiple signatures, and signatures of parts of the document.

3. The tradition digital signature returns a string primitive or binary data.

4. The traditional digital signature sends the signature value and the original message to the verifier for authentication, which affects the capacity of the network and affects the efficiency of signature confirmation.

5. The traditional digital signature uses the X.509 certificate, to express the type and value of the signature key.
On the other hand, the XML digital signature solves the above limitations by the following:

1. The size of the XML signature is very small, and it applies the digital signatures to element-level granularity.

2. The XML signature provides a means to sign the entire document, parts of a document, or multiple signatures written in the same document.

3. The XML signature returns a <Signature>, which is an XML element.

4. The XML signature is suitable for the distributed network.

5. The XML signature uses the <KeyInfo> to present easily and explicitly all the information of the signature key.

### 3.2 XML ENCRYPTION

XML encryption is a specification developed by the W3C Consortium (Imamura et al 2013). XML encryption (Selkirk 2001b, Ardagna et al 2007, and Yue-sheng et al 2010) describes a process for encrypting and decrypting data, and representing the result of the encryption, encryption algorithm, and encryption key, using the syntax of XML. The input data to the XML encryption process is an XML document, an XML element, an XML element contents, or non-XML data, such as binary data.

The main advantages of XML encryption is that, it provides the encryption of specific parts of an XML document, and multiple encryptions, which encrypt multiple parts of an XML document. XML encryption supports the confidentiality of the XML documents or parts of XML, documents by protecting the critical information, using cryptography algorithms. XML
encryption can be used to encrypt web pages, when there are neither authentications, access control mechanisms, nor an end-to-end encryption scheme. Moreover, the use of XML Encryption assists in solving the security problems, such as XML data eavesdropping and the safety of the encrypted documents during transmission and storing of them. XML encryption and signature are similar in:

1. The input, which is an XML document, parts of an XML document, or non-XML data (binary or digital data).

2. Both XML signature and encryption use the <KeyInfo>, which appears as a child of a <SignedInfo>, an <EncryptedData>, or an <EncryptedKey>. The <keyInfo> provides information to a recipient about the key, which will be used to validate signatures or decrypt encrypted data.

However, they differ in:

1. The output in the case of multiple encryptions or multiple signatures; for multiple encryptions, XML encryption returns a separate XML element, called an <EncryptedData> for each encryption in the same XML document, where the <EncryptedData> contains only one encrypted part. For multiple signatures, the XML signature returns one XML element, called <Signature> that contains all the signatures.

2. In XML encryption, the <KeyInfo> may contain a <KeyValue>, <KeyName>, or <RetrievalMethod>. However, in the XML signature, the <KeyInfo> contains either a <KeyValue>, <KeyName>, or <RetrievalMethod>. Only one of them appears in the <KeyInfo>, which is called the Required Choice.
3.2.1 XML Encryption Types

XML Encryption provides two types of encryption algorithms, namely, symmetric and asymmetric (Hashizume & Fernandez 2009, Shahgholi et al 2011, and Hao-yu et al 2011). The symmetric encryption algorithms, such as the Data Encryption Standard (DES) or Advanced Encryption Standard (AES) use a common key, called the symmetric (secret, single, or shared) key for both encryption and decryption.

In symmetric encryption, both the sender and receiver must know the same secret key. The sender uses the secret key to encrypt the plaintext and sends the ciphertext to the receiver. The receiver uses the same secret key to decrypt the message and returns the plaintext. Although the symmetric encryption is fast, the distribution of the key is a hard problem, because it requires a secure channel to exchange the secret key between the sender and receiver.

The asymmetric encryption algorithms, such as the RSA use the key pair (public key and private key) of the recipient. These public and private keys are linked mathematically, and the private key cannot be derived from its public key.

In asymmetric encryption, the sender encrypts a message using the receiver's public key, and the receiver uses his private key to decrypt the encrypted message. In asymmetric encryption, there is no need for key distribution, because the public keys are available to all the users, which makes the system strong. However, asymmetric encryption is slower than symmetric encryption.

When asymmetric and symmetric encryptions are used together, the encryption will be more effective. The symmetric key is used to encrypt the content, and then it is encrypted, using an asymmetric encryption algorithm.
Both the encrypted content and the encrypted symmetric key are sent to the recipient. The recipient will use his private key to decrypt the symmetric key, which itself is used to decrypt the content. This integration of the advantages of symmetric and asymmetric encryptions can conquer the key distribution problem of symmetric encryption, and the long time problem of asymmetric encryption. In both types of encryption, only recipients who own the shared key or the private key, that matches the public key used in the encryption process, can read the encrypted message after the decryption.

3.2.2 Syntax

The structure of the XML encryption document is depicted in Figure 3.6 (Hashizume & Fernandez 2009, Nordbotten 2009, Liu & Chen 2011, and Imamura et al 2013):

```
<EncryptedData Id? Type? MimeType? Encoding?>
  (<EncryptionMethod Algorithm />)?
  (<ds:KeyInfo>
    (<EncryptedKey>)?
    (<AgreementMethod >)?
    (<ds:KeyName>)?
    (<ds:RetrievalMethod URI Type? />)?
  <ds:/KeyInfo>)?
</CipherData>
  <CipherData>
    <CipherValue> | <CipherReference URI>
  </CipherData>
  (<EncryptionProperties Target? Id?>)?
</EncryptedData>
```

Figure 3.6 XML Encryption's Structure
In Figure 3.6, the <EncryptedData> is the root or result of XML encryption. When the encrypted data is an XML document, the <EncryptedData> becomes the root of the encrypted XML document. When the encrypted data is an XML element or element content, the <EncryptedData> is the result of XML encryption, because it replaces the element or content respectively, in the encrypted XML document. It presents the encryption content and the encryption keys of one resource. An XML document may contain zero or more <EncryptedData> elements, where each <EncryptedData> contains the encrypted data of only one resource. The <EncryptedData> cannot be a parent or a child of another <EncryptedData>.

The <EncryptedData> contains zero or one <EncryptionMethod>, zero or one <KeyInfo>, zero or one <EncryptionProperties>, and one <CipherData>. It has four optional attributes, namely, an Id, a Type, a MimeType, and an Encoding. The Id attribute allows it to be referenced by other encryptions or objects. The Type attribute identifies the type of the plaintext, which has been encrypted, such as Type = "element", Type = "element content", or Type = "attribute". The MimeType describes the media type of the encrypted data. The value of this attribute is a string, such as "image".

3.2.2.1 Encryption Method

The optional <EncryptionMethod> defines the encryption algorithm carried out on the resource to get the cipher data. It has a required attribute, called an Algorithm, which specifies the URI of the used algorithm in the W3C specification, such as the AES-128/256. If the element is omitted, the receiver must know the encryption algorithm; otherwise the decryption attempt will fail.
3.2.2.2 Key Information

The optional <ds:KeyInfo> includes the information to a recipient about the key (a private key of the recipient or shared key) used for decrypting the encrypted data in <EncryptedData> elements or encrypted symmetric keys in <EncryptedKey> elements. The <ds:KeyInfo> contains zero or one <ds:KeyValue>, zero or one <ds:KeyName>, zero or one <ds:RetrievalMethod>, or zero or one <EncryptedKey>.

The optional <ds:KeyValue> includes a plaintext key value (i.e., a secret key value in the case of symmetric encryption) needed for decrypting the encrypted data or encrypted keys. In asymmetric key encryption, the <ds:KeyValue> is not needed. In general, using the <ds:KeyValue> is not recommended. The optional <ds:KeyName> includes a name (a string value, and a whitespace is allowed) of the key used for decrypting the cipher data. It is necessary in asymmetric encryption to specify the name of the private key used for decrypting the cipher data. In symmetric encryption, since the receiver knows the shared key, it can be omitted.

The optional <ds:RetrievalMethod> is used to refer to an encrypted key information stored at another location outside an <EncryptedData>. It includes a URI attribute and a Type attribute. The required URI attribute is used to indicate the location of the encrypted key information. The optional Type attribute has the value "http://www.w3.org/2001/04/xmlenc#EncryptedKey". When an <EncryptedKey> is located outside an <EncryptedData>, the <ds:RetrievalMethod> with the attribute Type = "http://www.w3.org/2001/04/xmlenc#EncryptedKey" specifies the link of the <EncryptedKey> containing the encrypted symmetric key, required to decrypt the encrypted data in the <EncryptedData> or the encrypted key in another <EncryptedKey>. When the symmetric key is encrypted inside the
<ds:KeyInfo> of an <EncryptedData>, the <RetrievalMethod> is not necessary. Either or both the <ds:KeyName> and <ds:RetrievalMethod> are used to identify the same key.

The optional <EncryptedKey> includes an encrypted symmetric key used for decrypting the encrypted data or encrypted keys. It may be a stand-alone XML document, placed within an application document, or included as a child element of a <ds:KeyInfo> of an <EncryptedData>. When an <EncryptedKey> is a stand-alone XML document or placed within an application, a <ds:RetrievalMethod> with an attribute Type = "http://www.w3.org/2001/04/xmlenc#EncryptedKey" in a <ds:KeyInfo> in an <EncryptedData> is used to specify the link of the <EncryptedKey>. The <EncryptedKey> and <EncryptedData> are both derived from the same abstract type, namely, an <EncryptedType>.

In addition to all the child elements of the <EncryptedData>, the <EncryptedKey> contains zero or one <ReferenceList> and zero or one <CarriedKeyName>. A <CipherData> of an <EncryptedKey> includes the symmetric key in an encrypted form, while the <KeyInfo> within the <EncryptedKey> provides information about the key of the receiver used for decrypting the secret key (in general, the key is a receiver's private key).

The optional <ReferenceList> includes references to data or keys encrypted using this encrypted symmetric key. The <ReferenceList> may refer to multiple encrypted data and encrypted keys, which are encrypted by this encrypted symmetric key. The <ReferenceList> refers to the encrypted data, using a required element, called a <DataReference> and to the encrypted keys, using a required element, called a <KeyReference>. When the same encrypted symmetric key encrypts multiple <EncryptedData> or <EncryptedKey> elements, multiple <DataReference> or <KeyReference> elements occur inside the <ReferenceList>. 
The optional <CarriedKeyName> of an <EncryptedKey> includes the name of the encrypted symmetric key that has encrypted data or keys referenced in the <ReferenceList>. The whitespace is allowed in a <CarriedKeyName>. The name of a <CarriedKeyName> is similar to the name of a <ds:KeyName> within a <ds:KeyInfo> in an <EncryptedData>. When the same encrypted symmetric key encrypts multiple <EncryptedData> or <EncryptedKey> elements, the same <CarriedKeyName> may occur multiple times. The plaintext key values of all <EncryptedKey> elements, which have the same <CarriedKeyName> within a single XML document, are the same.

3.2.2.3 Cipher Data

The required <CipherData> of the <EncryptedData> includes the encrypted data. The encrypted data may be included explicitly or referenced, if it is stored in an external location. The <CipherData> contains either one <CipherValue> or one <CipherReference> (Required Choice). The <CipherValue> includes the encrypted data directly. The <CipherReference> refers to the encrypted data, if it is stored in an external location. The <CipherReference> includes a required URI attribute used to indicate the location of the encrypted data.

3.2.2.4 Encryption Property

The optional <EncryptionProperty> associates properties with the <EncryptedData> or <EncryptedKey>. It may include information, such as the date/time of the encryption process, and the type of the encrypted object. It includes an optional Target attribute, which identifies whether the type of an encrypted object is data or a key.

Figures 3.8, 3.9, 3.10, and 3.11 present examples of an XML encryption's structure:
Figure 3.7 XML Document for Students' Courses

Figure 3.7 presents the courses and grades of students. Because the grade information is sensitive, the <Course> is encrypted, and is depicted in Figure 3.8:

```xml
<? xml version = "1.0"?>
<Student>
    <Name> Abd El-Aziz </Name>
    <Semester>
        <Number> I </Number>
        <EncryptedData
            Type = "http://www.w3.org/2001/04/xmlenc#Element"
            xmlns = "http://www.w3.org/2001/04/xmlenc#">
            <EncryptionMethod
                Algorithm = "http://www.w3.org/2001/04/xmlenc#
                tripledes-cbc"/>
            <ds:KeyInfo
                xmlns:ds = "http://www.w3.org/2000/09/xmldsig#">
                <ds:KeyName> Omar </ds:KeyName>
            </ds:KeyInfo>
            <CipherData>
                <CipherValue> B34C5ACD6 </CipherValue>
            </CipherData>
        </EncryptedData>
    </Semester>
</Student>
```

Figure 3.8 Encryption of the Course Element
In Figure 3.8, the Triple DES, which is a symmetric algorithm encrypts the <Course>; the name of the symmetric key is Omar (i.e., the Key with the name Omar is used for the encryption and decryption), and the cipher value is B34C5ACD6. The encryption of the <Course>’s contents, which are of type Content is depicted in Figure 3.9:

```xml
<?xml version="1.0"?>
<Student>
  <Name> Abd El-Aziz </Name>
  <Semester>
    <Number> I </Number>
    <Course>
      <EncryptedData
        Type = "http://www.w3.org/2001/04/xmlenc#Content"
        xmlns = "http://www.w3.org/2001/04/xmlenc#">
        <EncryptionMethod
          Algorithm = "http://www.w3.org/2001/04/xmlenc#
                       tripledes-cbc"/>
        <ds:KeyInfo
          xmlns:ds = "http://www.w3.org/2000/09/xmldsig#">
          <ds:KeyName> Omar </ds:KeyName>
          </ds:KeyInfo>
        <CipherData>
          <CipherValue> AC3F4C5AC </CipherValue>
          </CipherData>
        </EncryptedData>
      </Course>
    </Semester>
  </Student>
```

Figure 3.9 Encryption of the Course Element’s Contents
The encryption of the `<Grade>`'s contents, which are character values, is depicted in Figure 3.10:

```xml
<?xml version="1.0"?>
<Student>
  <Name> Abd El-Aziz </Name>
  <Semester>
    <Number> I </Number>
    <Course>
      <Name> XML security </Name>
    </Course>
    <Grade>
      <EncryptedData
        xmlns="http://www.w3.org/2001/04/xmlenc#"
        xmlns:xmlenc="http://www.w3.org/2001/04/xmlenc#">
        <EncryptionMethod
          Algorithm="http://www.w3.org/2001/04/xmlenc#
          tripledes-cbc"/>
        <ds:KeyInfo
          xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
          <ds:KeyName> Omar </ds:KeyName>
        </ds:KeyInfo>
        <CipherData>
          <CipherValue> AC3GRTY45C </CipherValue>
        </CipherData>
      </EncryptedData>
    </Grade>
  </Semester>
</Student>
```

Figure 3.10 Encryption of the Grade Element's Contents
When an asymmetric encryption algorithm encrypts the symmetric key used to encrypt the grade value, XML encryption will be given, as in Figure 3.11:

```xml
<? xml version = "1.0 " ?>
<Student>
    <Name> Abd El-Aziz </Name>
    <Semester>
        <Number> I </Number>
    <Course>
        <Name> XML security </Name>
        <Grade>
            <EncryptedData Id = "ED"
                Type = "http://www.w3.org/2001/04/xmlenc#Content"
                xmlns = "http://www.w3.org/2001/04/xmlenc#"
            <EncryptionMethod
                Algorithm = "http://www.w3.org/2001/04/xmlenc#aes128-cbc"/>
            <ds:KeyInfo
                xmlns:ds = "http://www.w3.org/2000/09/xmldsig#"
            <RetrievalMethod URI = "#EK"
                Type = "http://www.w3.org/2001/04/xmlenc
                    #EncryptedKey"/>
                <ds:KeyName> Omar </ds:KeyName>
            </ds:KeyInfo>
        </CipherData>
        <CipherValue> AC3GRTY45C </CipherValue>
    </EncryptedData>
</EncryptedKey
    <EncryptedKey Id = "EK"

Figure 3.11 (Continued)
In Figure 3.11, the content of the <Grade> is encrypted with a symmetric key associated with the name Omar. The symmetric key is encrypted using the RSA, which is an asymmetric encryption algorithm, and it will be decrypted, using the private key associated with the name Ahmed. The <ReferenceList> refers to the <EncryptedData Id = "ED"> encrypted using the symmetric key, named Omar. The reference is done using the <DataReference>. The <CarriedKeyName> points to the symmetric key
Omar used to encrypt the data in the `<EncryptedData Id = "ED">`. Since the symmetric key, named Omar is used for the encryption and decryption, the value of the `<CarriedKeyName>` is similar to the value of the `<KeyName>`.

### 3.2.3 Processing and Implementations

In this work, the XML documents are encrypted by applying the following steps (Liu & Chen 2011):

1. Select an algorithm for the encryption and assign the used encrypting algorithm to an Algorithm attribute of an `<EncryptionMethod>`.

2. Generate a random key. In asymmetric encryption, the key pair is a public key for the encryption and a private key of a recipient for the decryption. In symmetric encryption, the key is a secret key. If the decryption keys (the private key of the recipient or secret key) are identified by a name or a URI, construct for them a `<ds:KeyInfo>`, which may contain a `<ds:KeyName>`, `<ds:KeyValue>`, or `<ds:RetrievalMethod>`. If an asymmetric algorithm is to encrypt the symmetric key, construct an `<EncryptedKey>` by recursively applying this encryption process. The `<EncryptedKey>` may be a child of the `<ds:KeyInfo>`, or it may be stored independently and be identified by a `<ds:RetrievalMethod>` with an attribute Type = "http://www.w3.org/2001/04/xmlenc#EncryptedKey".

3. Encrypt the data using the selected (symmetric/asymmetric) algorithm and the generated secret/public key.
4. Store the encrypted data either in a <CipherValue> or externally, and refer to the encrypted data by a <CipherReference>.

5. Construct a <CipherData> from the <CipherValue> or <CipherReference>.

6. Add the <EncryptionMethod>, <ds:KeyInfo> (optional), and <CipherData> to an <EncryptedData>, which is the final encryption result.

The following steps decrypt the encrypted data:

1. Investigate the <EncryptedData> to determine the used encryption algorithm, and the <ds:KeyInfo> to determine the decryption key (the private key of the recipient or secret key) that will be used for decrypting the cipher data.

2. If the decryption key is an encrypted symmetric key, investigate the <ds:KeyInfo> of the <EncryptedKey> to determine the decryption key (in general, it is the private key of the recipient) that will be used for decrypting the encrypted symmetric key.

3. Decrypt the data, which is contained in the <CipherData>.

Many library function tools assist researchers to implement the XML encryption. They are (Haron et al 2010):

1. Apache XML Security Library: Apache XML security (http://santuario.apache.org) is an open source initiative for XML security launched under Apache Software license. It is
developed in C++ and Java and supports the W3C specifications for XML encryption.

2. XML Security Library: The XML Security Library or known as the XMLSec library (Sanin et al 2013) is an open source initiative to enable the integration of XML security inside a third party solution. It is developed in C++, launched under MIT license, and supports the W3C specifications for XML encryption in both Windows and Linux operating systems.

3. XML Security Tool Application: The XML security tool plugin (Schadow 2005) for Eclipse allows the users to exploit the XML security features. This application is released under Eclipse Public License and is intended to bridge the lack of user acceptance in the XML technology. The XML interface is built on top of the Apache XML Security library using Java.


3.2.4 Comparing with Traditional Encryption

The limitations of traditional encryption are as follows: (Yue-sheng et al 2010):

1. The traditional encryption algorithms apply the encryption to data using a single key.
2. The SSL provides Point-to-Point security. Therefore, it is not suitable for the transmission modes of web services, such as the TCP, the FTP, and messages formation.

3. The SSL can execute the encryption for the complete information, but it cannot do it for the partial information.

On the other hand, XML encryption solves the above limitations by the following:

1. The XML encryption algorithm uses many symmetrical or asymmetrical keys to achieve the element level encryption.

2. XML encryption provides End-to-End security for the application procedure. Hence, it is flexible for the applications, especially, of mobile web services that demand more flexible, customizable, and better-optimized security schemes.

3. XML encryption provides the encryption of specific parts of an XML document and multiple encryptions, which encrypts multiple parts of an XML document.

3.3 SUMMARY

This chapter shows the background information about the XML signature and encryption in detail. For the XML signature, it explains the XML signature types, syntax of the XML signature, and canonicalization. Moreover, it shows the steps of the XML signature and verification processes, lists some of the function tools for the XML signature, and compares the XML and traditional signatures.
For XML encryption, it explains the XML encryption types, syntax of XML encryption. In addition, it shows the steps of the XML encryption and decryption processes, lists some of the function tools for XML encryption, and compares the XML and traditional encryption processes.
CHAPTER 4
EXTENSIBLE ACCES CONTROL MARKUP LANGUAGE

XACML is an XML-based language for access control that has been developed by OASIS (Rissanen 2013). XACML defines both an access control policy language, and a request and response language.

The policy language is an ABAC mechanism, used to construct expressions that make up an access control policy, which specifies the security mechanisms. In other words, the policy language defines the required constraints and conditions to a subject for accessing a resource, and carries out an action through a specific environment. The policy language is an extensible, flexible, highly expressive, standards-based, and general-purpose language. Moreover, it enables the specification of fine-grained policies, used to access the control to resources.

The request and response language describes the subjects making requests for accessing resources, and renders the authorization decisions granting or denying the access request. XACML is used not only for controlling the access to XML documents, but also for controlling the access to any type of resources.

XACML defines four logical modules:

1. Policy Administration Point (PAP) creates policies or policy sets, and manages, and stores them in an appropriate repository.
2. Policy Decision Point (PDP) evaluates an applicable policy and conveys an authorization decision.

3. Policy Enforcement Point (PEP) carries on an access control by conveying the decision requests to a PDP, and enforcing the authorization decisions returned by the PDP.

4. Policy Information Point (PIP) acts as the source of the required attribute values of a subject, a resource, an action, and an environment for a policy's evaluation.

The main functions offered by XACML can be summarized as follows (Ardagna et al 2007, Ardagna et al 2009):

1. **Policy combination**: XACML provides a method for combining policies specified independently. Therefore, different entities can define their policies on the same resource. Hence, when an access request on that resource is submitted, the system takes into account all the applicable policies.

2. **Combining algorithms**: XACML provides different combining algorithms; each one provides a way of reconciling multiple decisions into a single decision.

3. **Attribute-based restrictions**: XACML provides the definition of policies based on the attributes of subjects (e.g., name and address) and resources (e.g., creation date and type). Moreover, it includes built-in operators for comparing the attribute values and provides a method for adding nonstandard functions.
4. **Policy distribution**: Policies can be defined by different parties and carried out at different enforcement points. Moreover, XACML allows one policy to contain another one or refers to it.

5. **Implementation independence**: XACML provides an abstraction layer that insulates the policy-writer from its implementation details. This insulation guarantees that different implementations are executed consistently.

6. **Obligations**: XACML provides a method for specifying actions, called obligations, which must be executed in conjunction with the applicable policies that have decisions.

The main concepts of all XACML policies are a `<PolicySet>`, `<Policy>`, and `<Rule>`, which represent a single access control policy. The root element of each XACML policy document is exactly one `<PolicySet>`, `<Policy>`, or `<Rule>`. The `<Policy>` consists of one or more `<Rule>` elements. A `<policy>` has at least one `<Rule>`. The `<Rule>` includes the core logic of an XACML policy. The decision logic of the rules is included in a `<Condition>`. The `<Condition>` is a Boolean function that refines the applicability of the rule. If the `<Condition>` returns true, then the rule's Effect (Permit or Deny) is returned. If the `<Condition>` returns false, the PDP returns to the PEP the value NotApplicable.

If many `<Policy>` elements are contained in a `<PolicySet>`, the PDP needs a way to reconcile the effects returned by all policies. Hence, the concept of the Policy Combining Algorithm is introduced in a `<PolicySet>`. The final decision value of the policy-combining algorithm is called the authorization decision. Similarly, if many `<Rule>` elements are contained in a `<Policy>`, the PDP needs a way to reconcile the effects returned by all rules.
Hence, the concept of the Rule Combining Algorithm is introduced in each <Policy>.

Moreover, XACML provides a feature, called a <Target>, which is a set of simplified attribute values for a subject, a resource, an action, and an environment that must be satisfied for a <PolicySet>, <Policy>, or <Rule> to be applicable to a given request. If all the attribute values of a <Target> are satisfied with the associated <PolicySet>, <Policy>, or <Rule>, then the associated <PolicySet>, <Policy>, or <Rule> applies to the request (Scaglioso et al 2008, Hsieh et al 2010).

4.1 XACML POLICY LANGUAGE

XACML policies are XML documents with one <PolicySet>, one <Policy>, or one <Rule> as the root element. A <PolicySet> contains zero or more <PolicySet> elements (optionally), zero or more <Policy> elements (optionally), one <Target> (required), one identifier for the policy-combining algorithm (required), zero or one <ObligationExpressions> (optionally), and zero or one <AdviceExpressions> (optionally). A <Policy> contains one or more <Rule> elements (at least one <Rule>), one identifier for the rule-combining algorithm, one <Target>, zero or one <ObligationExpressions>, and zero or one <AdviceExpressions>. A <Rule> contains zero or one <Target>, zero or one <Condition>, one Effect attribute, zero or one <ObligationExpressions>, and zero or one <AdviceExpressions>. A <Target> is a set of attribute values to identify uniquely a subject, a resource, an action, and an environment that must be satisfied for a <PolicySet>, <Policy>, or <Rule> to be applicable to a given request. If all the conditions of a <Target> are satisfied with the associated <PolicySet>, <Policy>, or <Rule>, then the associated <PolicySet>, <Policy>, or <Rule> applies to the request. The <Target> must appear as a child of a <PolicySet> and <Policy>. However, it

4.1.1 Rule

A <Rule> contains a RuleId attribute as an identifier, zero or one <Description>, zero or one <Target>, zero or one <Condition>, one Effect attribute, zero or one <ObligationExpressions>, and zero or one <AdviceExpressions>. The required RuleId attribute is a string that assigns a unique name to the <Rule>. The optional <Description> contains a free description to the rule.

The optional <Target> defines the set of attribute values of the requests to which the rule is proposed to apply in the form of a logical expression on the attributes of the request. If the matches defined by the target are satisfied by the attributes of the request, the rule is applicable to the request. The rule is proposed to be applied to all entities of a particular data type, if this entity is omitted from the <Target>. If the <Target> is omitted from a <Rule>, the target of the <Rule> will be the same as the <Target> of its parent <Policy>.

The required Effect attribute of the rule indicates the rule consequence of a True evaluation for the rule. The Effect attribute value is Permit or Deny.

The optional <Condition> is a Boolean expression that must be satisfied (be true) for the rule to be assigned its Effect attribute value. It refines the applicability of the rule. For example, in the sentence "Only allow logins from 10 am to 6 pm", the condition indicates that, the access is allowed only in the interval [10 am - 6 pm]. If the <Condition> is omitted or evaluates to true, the condition value will be True. The condition value will be False if
the `<Condition>` evaluates to false. The condition value will be Indeterminate, if an operational error occurred during the evaluation, such as missing attributes, network errors while retrieving rules, division by zero, or syntax errors in the decision request or in the rule. Therefore, the `<Rule>` is evaluated as follows:

1. (If the `<Rule>` has not a `<Target>`, or the `<Target>` matches the attributes of the request) and the `<Condition>` evaluates to true, the rule value will be the value of the Effect attribute (Permit or Deny).

2. (If the `<Rule>` has not a `<Target>`, or the `<Target>` matches the attributes of the request) and the `<Condition>` evaluates to false, the rule value will be NotApplicable.

3. (If the `<Rule>` has not a `<Target>`, or the `<Target>` matches the attributes of the request) and the `<Condition>` evaluates to Indeterminate, the rule value will be Indeterminate `{P}`, if the Effect attribute value is Permit, or Indeterminate `{D}`, if the Effect attribute value is Deny. Indeterminate `{P}` means an Indeterminate from a policy or rule, which will be evaluated to Permit but not Deny. Indeterminate `{D}` means an Indeterminate from a policy or rule, which will be evaluated to Deny but not Permit.

4. If the `<Target>` of the `<Rule>` does not match the attributes of the request, the rule value will be NotApplicable. The Rule's condition value will not be considered.

5. If the `<Target>` of the `<Rule>` matching evaluates to Indeterminate, the rule's value will be Indeterminate `{P}`, if the Effect attribute value is Permit, or Indeterminate `{D}`, if
the Effect attribute value is Deny. The rule's condition value will not be considered.

4.1.1.1 Expression

The `<Condition>` contains one `<Expression>`. The `<Expression>` must return a value of type "http://www.w3.org/2001/XMLSchema#boolean". The `<Expression>` may be an `<AttributeDesignator>`, an `<AttributeSelector>`, an `<Apply>`, an `<AttributeValue>`, or a `<Function>`.

The `<AttributeDesignator>` is used as a pointer to identify a bag of all the values of the named attributes in the request subject by its category, identifier, and data type. A named attribute is the matched attribute of the request subject in the values of Category, AttributeId, and DataType attributes. Therefore, the `<AttributeDesignator>` may be used to retrieve one or more attribute values from the `<Attributes>` of the request subject context.

The `<AttributeSelector>` is used to retrieve a bag of unnamed and uncategorized attribute values from the `<Content>` of the request resource context. The values are retrieved from the nodes selected by applying an XPath expression to the XML content of the `<Content>` of the request resource context. Since both the `<AttributeDesignator>` and `<AttributeSelector>` may return multiple values, XACML provides an attribute type, called bag, which is an unordered collection that can contain duplicate values for a particular attribute.

The `<Apply>` represents a function and its arguments. The `<Apply>` contains a required FunctionId identifier that determines the function that will be applied to the arguments of the `<Apply>`, zero or more `<Expression>` elements that represent the arguments of the function, and zero or one `<Description>`, which is a free description to the `<Apply>`.
The <Function> is used to assign a name to a function, which is an argument to another function defined by its parent <Apply>. It contains a required FunctionId identifier.

The <AttributeValue> includes a literal attribute value, and it has a required DataType attribute that represents the data type of the literal attribute value.

4.1.1.2 Obligation Expressions

An obligation specifies an action described by the attribute values and must be executed, when the result of the rule is Permit/Deny. Only rules that are evaluated and have an effect of permit or deny, can return obligations. For example, an obligation states that, "all accesses to an employee's data have to be logged". Another obligation states, "send an email to the administrator when the actual resource is accessed". The <Rule> specifies the obligation, using an optional <ObligationExpressions>. This element contains one or more <ObligationExpression> elements.

The <ObligationExpression> contains a required ObligationId attribute, FulfillOn attribute, and zero or more <AttributeAssignmentExpression> elements.

The required ObligationId attribute is an obligation identifier. The required FulfillOn attribute determines for which effect (Permit/Deny) this obligation must be carried out by the PEP.

The <AttributeAssignmentExpression> retrieves the attribute values that describe the action specified by the obligation. Each retrieved attribute value will be included in an <AttributeAssignment> of an <Obligation> within the response. If the evaluation of the
<AttributeAssignmentExpression> retrieves an empty bag of values, there is no corresponding <AttributeAssignment> in the <Obligation> of the response. The <AttributeAssignmentExpression> contains a required AttributeId attribute, an optional Category attribute, and one <Expression>.

The AttributeId is the value of the AttributeId attribute for the retrieved attribute values included in the corresponding <AttributeAssignment> of the <Obligation> within the response.

The Category attribute indicates the category of the retrieved attribute values, which may be a subject category or a resource category. Hence, the category attribute is used to distinguish among the attribute values retrieved from the request context. The value of the Category attribute in the <AttributeAssignmentExpression> must be equal to the value of the Category attribute in the corresponding <AttributeAssignment>.

The required <Expression> retrieves the required attribute values for an obligation using an <AttributeDesignator>, <AttributeSelector>, or <AttributeValue>. Therefore, when a PDP evaluates a rule, which has an effect of permit or deny, and the effect matches the value of the FulfillOn attribute contained in an <ObligationExpression>, the PDP evaluates the expressions in the <AttributeAssignmentExpression>, includes the retrieved attribute values in an <Obligation>, and returns an <Obligations>, which is containing the <Obligation>, within the response context to the PEP.

4.1.1.3 Advice Expressions

In addition to the obligation, the <Rule> may contain an advice that specifies an action described by the attribute values, and must be executed when the result of the rule is Permit/Deny. Only rules that are evaluated and have an effect of permit or deny can return advices. The <Rule> specifies the
advice using an optional <AdviceExpressions>. This element contains one or more <AdviceExpression> elements.

The <AdviceExpression> contains a required AdviceId attribute, AppliesTo attribute, and zero or more <AttributessignmentExpression> elements.

The required AdviceId attribute is an advice identifier. The required AppliesTo attribute determines for which effect (Permit/Deny) this advice must be carried out by the PEP.

The <AttributeAssignmentExpression> retrieves the attribute values that describe the action specified by the advice. Each retrieved attribute value will be included in an <AttributeAssignment> of an <Advice> within the response. If the evaluation of the <AttributeAssignmentExpression> retrieves an empty bag of values, there is no corresponding <AttributeAssignment> in the <Advice> of the response. The <AttributeAssignmentExpression> contains a required AttributeId attribute, an optional Category attribute, and one <Expression>.

The AttributeId is the value of the AttributeId attribute for the retrieved attribute values included in the corresponding <AttributeAssignment> of the <Advice> within the response.

The Category attribute indicates the category of the retrieved attribute values, which may be a subject category or a resource category. Hence, the category attribute is used to distinguish among the attribute values retrieved from the request context. The value of the Category attribute in the <AttributeAssignmentExpression> must be equal to the value of the Category attribute in the corresponding <AttributeAssignment>.
The required <Expression> retrieves the required attribute values for the advice using an <AttributeDesignator>, <AttributeSelector>, or <AttributeValue>. Therefore, when a PDP evaluates a rule that has an effect of permit or deny, and the effect matches the value of the AppliesTo attribute contained in an <AdviceExpression>, the PDP evaluates the expressions in the <AttributeAssignmentExpression>, includes the retrieved attribute values in an <Advice>, and returns an <AssociatedAdvice>, which is containing the <Advice>, within the response context to the PEP. Contrary to the obligation, an advice may be safely ignored by the PEP (Rissanen 2013).

4.1.2 Policy

A <Policy> contains a PolicyId attribute as an identifier, zero or one <Description>, one or more <Rule> elements, an identifier for the rule-combining algorithm, one <Target>, zero or one <ObligationExpressions>, and zero or one <AdviceExpressions>. The required PolicyId attribute assigns a unique name to the <Policy>. The optional <Description> contains a free description to the policy.

The policy may contain an optional <ObligationExpressions> and <AdviceExpressions> that are explained in the subsection 4.1.1. Only policies that are evaluated and have a decision of permit or deny, can return obligations or advices. When a PDP evaluates a policy that has an authorization decision (permit or deny), and the decision matches the value of the FulfillOn or AppliesTo attribute contained in an <ObligationExpression> or <AdviceExpression> respectively, it evaluates the expressions in the <AttributeAssignmentExpression>, includes the retrieved attribute values in an <Obligation> or <Advice>, and returns an <Obligations> containing the <Obligation>, or an <AssociatedAdvice> containing the <Advice>, within the response context to the PEP (Rissanen 2013).
4.1.2.1 Target

The writer of a Policy or PolicySet may define the required <Target> of the <Policy> or PolicySet, or the <Target> may be calculated from the <Target> of its inner components PolicySet, Policy, or Rule. If the policy's writer defines the <Target> of a <Policy>, any <Rule>, contained in the <Policy> and has the same <Target> as its parent <Policy>, may omit its <Target>. These <Rule> elements inherit the <Target> of their parent <Policy>. If the <Target> needs to be calculated, there are two logical ways that can be used to calculate the <Target> of the <Policy> or PolicySet.

In the first method, the <Target> of the root <Policy> or PolicySet is calculated as the union of all the <Target> elements of the inner components (children of the root) PolicySet, Policy, or Rule. In this case, the <Target> of the root will be applicable to any decision request that matches the <Target> of at least one child element.

In the second method, the <Target> of the root <Policy> or PolicySet is calculated as the intersection of all the <Target> elements of the inner components PolicySet, Policy, or Rule. In this case, the <Target> of the root will be applicable only to a decision request that matches the <Target> of every child element of the root.

4.1.2.2 Rule Combining Algorithms

The required rule-combining algorithm identifier explains how the results of the rules' evaluations are reconciled to evaluate the policy; i.e., it specifies the value of the authorization decision (policy's value) that is placed in the response context by the PDP.
XACML defines different combining algorithms, namely, Deny overrides algorithm, Permit overrides algorithm, First applicable algorithm, and Only-one-applicable algorithm. The Deny overrides algorithm states that:

1. If there is a rule that evaluates to Deny, the policy evaluates to Deny.
2. If all the rules evaluate to NotApplicable, the policy evaluates to NotApplicable.
3. If all the rules evaluate to Permit, the policy evaluates to Permit.
4. If some rules evaluate to Permit and some evaluate to NotApplicable, the policy evaluates to Permit.

The Permit overrides algorithm states that:

1. If there is a rule that evaluates to Permit, the policy evaluates to Permit.
2. If all the rules evaluate to NotApplicable, the policy evaluates to NotApplicable.
3. If all the rules evaluate to Deny, the policy evaluates to Deny.
4. If some rules evaluate to Deny and some evaluate to NotApplicable, the policy evaluates to Deny.

The First applicable algorithm states that:

1. The rules are evaluated in the listing order, in which they appear in the <Policy>.
2. For each rule, if the rule's target matches the request's target and the condition evaluates to true, the policy evaluates to the rule's Effect attribute value (Permit or Deny), for the selected rule.

3. If the target evaluates to false or the condition evaluates to false, evaluate the next rule in the order.

4. If there is no rule in the order, the policy evaluates to NotApplicable.

5. If an error occurs during the evaluation of the target or condition of a rule, stop the rules' evaluation, and the policy evaluates to Indeterminate.

The Only-one-applicable algorithm states that:

1. If there is more than one rule, which are applicable by their target, the policy evaluates to Indeterminate.

2. If there is no applicable rule by its target, the policy evaluates to NotApplicable.

3. If only one rule is applicable by its target, the policy result is the same as the evaluation result of that rule.

4. If an error occurs during the evaluation of a rule, the policy evaluates to Indeterminate.

According to the selected combining algorithm, the authorization decision returned to the PEP can be Permit, Deny, NotApplicable (when no applicable rules could be found), or Indeterminate (when some errors occurred during the access control process) (Ardagna 2007, Rissanen 2013).
4.1.3 Policy Set

A <PolicySet> contains a PolicySetId attribute as an identifier, zero or one <Description>, zero or more <PolicySet> elements, zero or more <Policy> elements, one <Target>, an identifier for the policy-combining algorithm, zero or one <ObligationExpressions>, and zero or one <AdviceExpressions>.

The required PolicySetId attribute assigns a unique name to the <PolicySet>. The optional <Description> contains a free description to the policy set. The <Policy> and <Target> are presented above in the subsection 4.1.2. The required policy-combining algorithm identifier explains how the results of the policies' evaluations are reconciled together to evaluate the policy set; i.e., it specifies the decision value (policy set's value) placed in the response context by the PDP. The policy-combining algorithms are similar to the rule-combining algorithms that are discussed above.

Only policy sets that are evaluated and have a decision of permit or deny, can return obligations or advices. When a PDP evaluates a policy set that has an authorization decision (permit or deny), and the decision matches the value of the FulfillOn or AppliesTo attribute contained in an <ObligationExpression> or <AdviceExpression> respectively, it evaluates the expressions in the <AttributeAssignmentExpression>, includes the retrieved attribute values in an <Obligation> or <Advice>, and returns an <Obligations> containing the <Obligation>, or an <AssociatedAdvice> containing the <Advice>, within the response context to the PEP. An example of an XACML policy set document is depicted in Figure 4.1:
<PolicySet PolicySetId = "pls-0001" PolicyCombiningAlgId = "urn:oasis:names:tc:xacml:3.0:policy-combining-algorithm:deny-overrides">
  <Description>
    It is the description of the policy set
  </Description>
  <Target>
    <Subjects>
      <Subject>
        <SubjectMatch  MatchId = "urn:oasis:names:tc:xacml:1.0:function:string-equal">
          <AttributeValue  DataType = "http://www.w3.org/2001/XMLSchema#string">
            Omar
          </AttributeValue>
        </SubjectMatch>
      </Subject>
    </Subjects>
  </Target>
  <Policy PolicyId = "pol-0001" RuleCombiningAlgId = "urn:oasis:names:tc:xacml:3.0:rule-combining-algorithm:deny-overrides">
    <Description>
      It is the description of the policy
    </Description>
    <Rule  RuleId = "rul-0001" Effect = "Permit"/>
  </Policy>
</PolicySet>

Figure 4.1 Example of an XACML Policy Set Document
In Figure 4.1, the required attribute PolicySetId assigns a name to the <PolicySet>. The name of a <PolicySet>, <Policy>, or <Rule> has to be unique for a given PDP, so that there is no ambiguity if a <PolicySet>, <Policy>, or <Rule> is referenced from another <PolicySet>, <Policy>, or <Rule>.

The required identifier PolicyCombiningAlgId specifies the algorithm used to reconcile the results of the various <Policy> elements that may be in the <PolicySet>. The Deny-overrides policy-combining algorithm states that, if there is a policy that evaluates to Deny, the policy set evaluates to Deny. If all the policies evaluate to NotApplicable, the policy set evaluates to NotApplicable. If all the policies evaluate to Permit, the policy set evaluates to Permit. If some policies evaluate to Permit and some evaluate to NotApplicable, the policy set evaluates to Permit.

The optional <Description> has a text, which is a description of the <PolicySet>. The policy set's <Target> indicates that this policy set is applicable to any request from a subject, whose name is Omar, to execute any action on any resource. The attribute values in a given request are compared with the attribute values in the <Target>. If all the attribute values are matched together, the request is applicable. Hence, the request is further checked against the <Target> of the inner <Policy> and the <Target> of the inner <Rule> elements. If the attribute values of the request and those in the <Target> do not match, the request is not applicable. If an error occurred during the evaluation, such as missing attributes, network errors while retrieving policies, division by zero during a policy evaluation, or syntax errors in the decision request or in the policy, the request is indeterminate. If the <Target> is empty, the <PolicySet> is applicable to any access request.

The <Match> matches the attribute values in the <Attributes> of the request context with the embedded attribute value of the policy set's
<Target> in the <AttributeValue>. The required MatchId attribute specifies the matching function that will be used to compare the attribute values of the request and those of the <Target>. The value of this attribute must be of the type xs:anyURI. The required <AttributeValue> specifies the attribute data type, and the attribute value of the subject. The <AttributeDesignator> is used as a pointer to identify a bag of all the values of the named attributes in the request by its category, identifier, and data type. The <Match> must contain either the <AttributeDesignator> or <AttributeSelector> once, which is called the Required Choice; i.e., only one of them will appear once in the <Match>.

Moreover, in Figure 4.1, the required attribute PolicyId assigns a unique name to the <Policy>. Each <Policy> must have a unique identifier for a given PDP. The required identifier RuleCombiningAlgId specifies the algorithm that will be used to reconcile the results of the various <Rule> elements of the <Policy>. The Deny-overrides rule-combining algorithm states that, if there is a rule that evaluates to Deny, the policy evaluates to Deny. If all the rules evaluate to NotApplicable, the policy evaluates to NotApplicable. If all the rules evaluate to Permit, the policy evaluates to Permit. If some rules evaluate to Permit and some evaluate to NotApplicable, the policy evaluates to Permit.

The policy's <Target> describes the decision requests to which this policy applies. If the attribute values in a decision request do not match the attribute values specified in the policy's <Target>, the remainder of the policy (the <Rule> elements of the <Policy>) does not need to be evaluated and the policy result is NotApplicable. If the <Target> is empty, the policy is applicable to any decision request.

The optional <Description> has a text, which is a description of the <Policy>. The <Policy> has only one <Rule>. The required attribute RuleId assigns a unique name to the <Rule>. Each <Rule> must have a unique
identifier for a given PDP. The required Effect attribute of the <Rule> specifies that, the effect of this rule will be Permit when the rule evaluates to True; i.e., the access request should be permitted. If the rule evaluates to False, the rule result will be NotApplicable. If an error occurs during the evaluation of the rule, the rule result will be Indeterminate.

The <Target> of a <Rule> describes the decision requests to which this rule applies. If the attribute values in a decision request do not match the attribute values specified in the rule's <Target>, the remainder of the rule (the <Condition>) does not need to be evaluated, and the rule's value will be NotApplicable. If the <Target> is empty, the rule is applicable to any decision request. Since the <Rule> in Figure 4.1 has not a <Target>, the <Rule> will inherit the empty <Target> of its parent <Policy>, and hence, it is applicable to any decision request. Since the <Rule> in Figure 4.1 has not a <Condition>, the rule's condition evaluates to True (Ardagna et al 2007, Xu et al 2011, and Rissanen 2013).

4.2 XACML REQUEST AND RESPONSE

In addition to the format of the policy language, XACML specifies the format of an authorization request and its response. This format is called the XACML Context. The Request Context is composed of a set of simplified attribute values for a <Subject>, a <Resource>, an <Action>, and an <Environment> that must be satisfied for a <Target> of a <PolicySet>, <Policy>, or <Rule>.

The <Subject> describes the person making the access request. The <Resource> describes the resource to which the subject has an access request. The <Action> specifies the actions that the subject needs to carry out on the resource. The <Environment> describes the resource environment using a set of attributes, which are relevant to take an authorization decision, and are
independent of a particular subject, resource, and action, such as date or time. For example, in the following access request, "Allow the head of a department of a faculty to update the information of the research scholars from the research scholars' web site on the faculty server". The subject is the head of the department, the target resource is the research scholars' web site on the faculty server, and the action is updating the information of the research scholars.

The <Request> contains one or more <Attributes> elements. Each <Subject>, <Resource>, <Action>, and <Environment> is associated with an <Attributes> to specify its attribute values. There may be multiple <Attributes> elements with different categories for each component. The required <Attributes> describes the attributes of the request context by using a set of <Attribute> elements associated with an attribute, called Category. The <Attributes> contains a Category attribute, an optional xml:id attribute, zero or more <Attribute> elements, and zero or one <Content>.

The Category attribute is a required attribute, which is used to specify the attribute category of the contained attributes. The Category attribute is used to distinguish among the attributes of a subject, a resource, an action, and an environment.

For a subject who needs the access request, the value of the category is "urn:oasis:names:tc:xacml:1.0:subject-category:access-subject". For a subject who receives the results of the request, the value of the category is "urn:oasis:names:tc:xacml:1.0:subject-category:recipient-subject". For a resource, the value of the category is "urn:oasis:names:tc:xacml:3.0:attribute-category:resource". For an action, the value of the category is "urn:oasis:names:tc:xacml:3.0:attribute-category:action". For an environment, the value of the category is "urn:oasis:names:tc:xacml:3.0:attribute-category:environment".
The optional xml:id attribute assigns a unique identifier to the <Attributes>. The optional <Content> includes the contents of a resource in a free XML document format. For example, the <Content> may include a part of an XML document, and the <AttributeValue> may include an xpath for this part of the XML document. The <AttributeSelector> may refer to the <Content>.

The optional <Attribute> specifies the data type and attribute values of the attributes, which belong to the category of the request. It contains a required AttributeId attribute, and one or more <AttributeValue> elements. The required AttributeId attribute assigns a unique identifier to the <Attribute>. The <AttributeDesignator> may refer to the <Attribute> using the AttributeId attribute.

In a <Subject>, when the value of the AttributeId is "urn:oasis:names:tc:xacml:1.0:subject:subject-id", it indicates that, the <AttributeValue> will contain the name of the subject. In a <Resource>, when the value of the AttributeId is "urn:oasis:names:tc:xacml:1.0:resource:resource-id", it indicates that, the <AttributeValue> will contain the resource to which access is requested. In an <Action>, when the value of the AttributeId is "urn:oasis:names:tc:xacml:1.0:action:action-id", it indicates that, the <AttributeValue> will contain the action for which access is requested. In an <Environment>, when the value of the AttributeId is "urn:oasis:names:tc:xacml:1.0:environment:current-dateTime", it indicates that, the <AttributeValue> will contain date and time.

The required <AttributeValue> includes the value of the attribute that belongs to the category of the request, and has a required attribute, called DataType. The content of the <AttributeValue> may be empty, occur once, or multiple times. An example of a request context is shown in Figure 4.2:
<Request>
  <Subject>
    <Attributes Category = "urn:oasis:names:tc:xacml:1.0:subject-category:access-subject">
      <Attribute
        AttributeId = "urn:oasis:names:tc:xacml:1.0:subject:subject-id">
        <AttributeValue
          DataType = "http://www.w3.org/2001/XMLSchema#string">
          Omar
        </AttributeValue>
      </Attribute>
    </Attributes>
  </Subject>
  <Resource>
    <Attributes Category = "urn:oasis:names:tc:xacml:3.0:attribute-category:resource">
      <Attribute
        AttributeId = "urn:oasis:names:tc:xacml:1.0:resource:resource-id">
        <AttributeValue
          DataType = "http://www.w3.org/2001/XMLSchema#string">
          /some/example.xml
        </AttributeValue>
      </Attribute>
    </Attributes>
  </Resource>
  <Action>
    <Attributes Category = "urn:oasis:names:tc:xacml:3.0:attribute-category:action">
      <Attribute
        AttributeId = "urn:oasis:names:tc:xacml:1.0:action:action-id">
      </Attribute>
    </Attributes>
  </Action>
</Request>

Figure 4.2 (Continued)
<AttributeValue>
   DataType = "http://www.w3.org/2001/XMLSchema#string">
    sign
</AttributeValue>
</Attribute>
</Attributes>
</Action>
</Environment>
<Attributes Category = "urn:oasis:names:tc:xacml:3.0:attribute-category:environment">
    <Attribute AttributeId = "urn:oasis:names:tc:xacml:1.0:environment:current-dateTime">
        <AttributeValue>
            DataType = "http://www.w3.org/2001/XMLSchema#dateTime">
                2010-06-14T11:12:03
            </AttributeValue>
        </Attribute>
    </Attributes>
</Environment>
</Request>

**Figure 4.2 Example of an XACML Request Document**

In Figure 4.2, the first <Attributes> contains the attributes of the subject, who is making the access request. The subject has only one attribute that indicates his name as Omar.

The second <Attributes> contains the attributes of the resource to which the subject Omar has made the access request. The resource is identified by the resource's URI (/some/example.xml).
The third `<Attributes>` indicates that, sign is the action that the subject wants to perform on the requested resource. When the PDP receives the request context in Figure 4.2, it will locate the policy set in Figure 4.1 in its policy set repository. Hence, it compares the attribute values in the request context with the attribute values in the policy set's target. Since the attribute values in the policy set's target match the attribute values in the request context, the PDP will continue the evaluation.

Now, the PDP compares the attribute values in the request context with the attribute values in the target of the policy. Since the policy's target is empty, the policy matches this request context. Finally, the PDP compares the attribute values in the request context with the attribute values in the target of the only rule in the policy. Since the rule's target is omitted, the rule inherits the target form its parent, which is the policy element. Therefore, the rule's target matches the request context. Hence, the policy set evaluates to Permit.

A Response Context includes the authorization decision, which is obtained from the evaluation of the decision request against the policy by the PDP. It contains a sequence of one or more results in the `<Result>`. The `<Response>` contains one or more `<Result>` elements.

The `<Result>` includes the authorization decision from the PDP. The `<Result>` contains one `<Decision>`, zero or one `<Status>`, zero or one `<Obligations>`, zero or one `<AssociatedAdvice>`, and zero or more `<Attributes>` elements.

The required `<Decision>` includes the following authorization decisions:

1. Permit, if the requested access is permitted.
2. Deny, if the requested access is denied.
3. Indeterminate, if some error occurred during a policy set, policy, or rule evaluation process, such as missing attributes, network errors while retrieving policies or rules, division by zero during a policy or rule evaluation, or syntax errors in the decision request, in the policy, or in the rule.

4. NotApplicable, if no applicable policies or rules could be found.

If errors occurred during the evaluation of the decision request, the optional <Status> describes these occurred errors, and includes the information about them optionally.

The optional <Obligations> includes a list of obligations that must be fulfilled by the PEP. The <Obligations> contains one or more <Obligation> elements. The <Obligation> is composed of a set of attribute values that describe the action defined by the obligation. It contains an ObligationId attribute and zero or more <AttributeAssignment> elements. The required ObligationId attribute is an identifier for the obligation. The optional <AttributeAssignment> includes an attribute value, which is an argument retrieved from an obligation expression. It has a required AttributeId as an Identifier and an optional Category attribute (Scaglioso et al 2008, Rissanen 2013). An example of a response context is shown in Figure 4.3:

```xml
<Response>
  <Result>
    <Description> Permit </Description>
  </Result>
</Response>
```

**Figure 4.3 Example of an XACML Response Document**
4.3 DATA FLOW OF THE XACML MODEL

The data flow through the XACML model can be summarized as follows (Rissanen 2013):

1. The PAP writes policies and policy sets and makes them available to the PDP.

2. The requester sends an access request to the PEP module, which must execute the access decision that will be taken by the PDP.

3. The PEP module sends the access request to the context handler in its original format. It includes optionally, attributes of the subject, resource, action, and environment.

4. The context handler converts the original request into a canonical format, called the XACML request context, and sends it to the PDP.

5. The PDP requests any additional attributes of the subject, resource, action, environment, and other categories from the context handler.

6. The context handler makes queries to the PIP module to get the additional attributes.

7. The PIP interacts with the subject, resource, action, and environment modules, and provides the attribute values about them to the context handler.

8. The context handler sends the XACML request to the PDP.
9. The PDP identifies the applicable policies by means of the PAP module, and retrieves the required attributes and, possibly, the resource from the context handler.

10. The PDP evaluates the policies, and returns the XACML response context to the context handler.

11. The context handler converts the XACML response context to the original format of the PEP, and returns it to the PEP together with an optional set of obligations.

12. If the response context is Permit or Deny, the PEP fulfils the set of obligations.

13. If the access is permitted, the PEP performs the access. Otherwise, it denies access.

4.4 IMPLEMENTATIONS OF XACML

Sun implements an open source implementation of XACML in Sun's XACML (2004). The implementation was written in the Java programming language. This implementation supports parsing both policy and request/response documents, determining the applicability of policies, and evaluating requests against policies. All the standards attribute types, functions, and combining algorithms are supported, and there are APIs for adding new functionalities, as needed.

XACML.NET, which is available in "http://mvpos.sourceforge.net/download.htm", provides a graphical editor, called Control Center that allows the creating, editing, and evaluating of the XACML policies and requests. In this tool, the PDP component has been implemented in a pure .Net code (C#). It conforms to the XACML 1.0 specification, launched by OASIS (Mourad et al 2011).
The UMU-XACML-Editor was released by the University of Murcia (UMU) to assist in the creation and validation of the XACML policies. It is a graphic editor for policy definition, and provides a syntax-directed manner to construct a correct XACML policy. It has been implemented in Java programming language. It has been used a tree-based user interface design to ease the policy integration. Hence, it allows the users to edit and save policies in a friendly user interface, with the constraints of the XACML 2.0 specifications that were defined by OASIS (Mourad et al 2011).

XACMLight, which is available at http://sourceforge.net/projects/xacmllight/, is Axis2web service that implements a PDP and PAP, and defines all functions that are defined by the XACML 2.0 specifications.

4.5 LIMITATIONS OF XACML

The following are the limitations of XACML:

1. Although XACML is an integral policy description language, the structure of the existing XACML policy is very complex and hard. Therefore, it is necessary for the users to understand XACML well, in order to write all the securing policy specifications (Lang et al 2008).

2. The flexibility and expressiveness of XACML causes its complexity and verbosity. It is very difficult to work directly with XACML, especially if XACML is used to express high-level access control concepts, such as the separation of duties (Alm & Illig 2010).

3. XACML suffers from complexity and difficulty, and expertise is needed for writing all policy specifications by hand, which
limits its potential in supporting the security of web services. Moreover, a developer should have an expert in the business logic, security, policy specification languages, and web service technologies, to define the complex XACML constructs, such as policies, rules, targets, subjects, and actions, which is a cumbersome problem (Mourad et al 2011).

4.6 RBAC PROFILE OF XACML

Access control is concerned with authorizing a user to fulfil a specific task on a resource, such as a file, database object, or service (Decker 2008). This authorization decision is based on the organization's security purposes. The most commonly used model for modeling security is the RBAC model (Sandhu et al 1996). RBAC uses users' roles and the rules that determine the allowed accesses for users in their roles, to control the access to an organization. RBAC determines and implements security policies, which are mapped to an organization's structure. The RBAC approach satisfies security for organizations as long as they grant access only according to the organizational roles.

The basic concept of the RBAC model is that, users are assigned to roles, permissions are assigned to roles, and hence, users obtain permissions by being members of roles. The relationship between users and roles is a many-to-many relation; i.e., a user can be assigned to many roles and a role can have many users. Similarly, the relationship between permissions and roles is a many-to-many relation; i.e., a permission can be assigned to many roles and a role can have many permissions. The RBAC model has four levels, namely, the flat RBAC, hierarchical RBAC, constrained RBAC, and symmetric RBAC. The flat RBAC contains the basic requirement for RBAC. It has the requirement for user-role assignment revision. The hierarchical
RBAC has a requirement for supporting role hierarchies. The constrained RBAC adds a requirement for implementing the Separation of Duties (SoD). The Separation of duties distributes the responsibilities of a task over multiple users. There are two types of separation of duties, namely, static separation of duty (based on user-role assignment), and dynamic separation of duty (based on user-role activation). The symmetric RBAC adds a requirement for permission-role revision, similar to the flat RBAC. The advantage of the RBAC model lies in the easy and effective administration of security policies (Sandhu et al 1996).

The RBAC cannot specify any constraints on the location of the user, and the suitable time of his request to access a resource. Therefore, to limit the access of resources according to the location of the user and the suitable time of his request, OASIS has defined the RBAC profile of XACML, to add functionalities to limit the time and the location of the requester (Aburahma & Stumptner 2009). The RBAC profile of XACML satisfies the requirements for core and hierarchical role based access control. This profile does not need any extensions to the standard XACML (Anderson 2005).

4.6.1 Role

A role represents a job function or title within an organization. In the RBAC profile of XACML, roles are described as Subject attribute values in Role <PolicySet> and Permission <PolicySet>, or as Resource attribute values in a Role Assignment <PolicySet> or <Policy> and in a HasPrivilegesOfRole <Policy>. The role is described in two ways. The first way is depicted in Figure 4.4:
Figure 4.4 First Way of Describing Roles

The second way is depicted in Figure 4.5:

Figure 4.5 Second Way of Describing Roles
4.6.2 Policy

In the RBAC profile of XACML, there are four policies:

1. **Role <PolicySet> (RPS):** is a <PolicySet> that represents a role attribute and its value in the <Subject> of its <Target>. It companions the role with a Permission <PolicySet> that contains the permissions of the role. Each Role <PolicySet> can refer to a single Permission <PolicySet>, but it cannot contain or refer to any other <Policy> or <PolicySet> elements.

2. **Permission <PolicySet> (PPS):** is a <PolicySet>, which includes the permissions of a particular role. It contains the <Policy> and <Rule> elements, which describe the allowed resources for the subjects, and the actions that they can carry out on the resources. Moreover, they describe the conditions that must be satisfied to allow the subjects to access the resources. A Permission <PolicySet> may inherit all the permissions associated with other roles that are junior to the given role by refereeing to different Permission <PolicySet> elements associated with other roles. If the Permission <PolicySet> has a <Target>, it must not restrict the subjects to which the <PolicySet> is applicable. The Permission <PolicySet> depends on its Role <PolicySet> to be sure that only subjects having the corresponding role attribute will have the permissions in the given Permission <PolicySet>.

3. **Role Assignment <Policy> or <PolicySet> (RAP):** is a <Policy> or <PolicySet> that assigns roles to subjects. A Role Enablement Authority (REA) uses this type of policy optionally.
4. HasPrivilegesOfRole <Policy>: is a <Policy> in a Permission <PolicySet> that checks whether a subject has the permissions associated with a given role or not.

A Permission <PolicySet> document must be stored in a way that it cannot be used as a policy for an XACML PDP. A Permission <PolicySet> is only ready through the corresponding Role <PolicySet> to be sure that, a Permission <PolicySet> must be allowed to every subject to provide hierarchical roles. Figures 4.6, 4.7, and 4.8 show examples of a Permission <PolicySet>, Role <PolicySet>, and Role Assignment <Policy> respectively (Anderson 2005):

```
<PolicySet xmlns="urn:oasis:names:tc:xacml:3.0:policy:schema:os"
PolicySetId = "PPS:manager:role" PolicyCombiningAlgId="&policy-combine; permit-overrides">
  <Policy PolicyId = "Permissions:for:the:manager:role"
    RuleCombiningAlgId = "&rule-combine; permit-overrides">
    <Rule RuleId = "Permission:to:update:salary file" Effect="Permit">
      <Target>
        <Resources>
          <Resource>
            <ResourceMatch MatchId = "&function; string-equal">
              <AttributeValue
                DataType ="&xml; string">salary file</AttributeValue>
            <ResourceAttributeDesignator
              AttributeId = "&resource; resource-id"
              DataType = "&xml; string"/>
            </ResourceMatch>
          </Resource>
        </Resources>
        <Actions>
          <Action>
```
Figure 4.6 Permission <PolicySet> for a Manager

```xml
<PolicySet xmlns="urn:oasis:names:tc:xacml:3.0:policy:schema:os"
          PolicySetId = "RPS:manager:role"
          PolicyCombiningAlgId = "&policy-combine; permit-overrides">
  <Target>
    <Subjects>
      <Subject>
        <SubjectMatch MatchId = "&function; string-equal">
          <AttributeValue
            DataType="&xml; string">
            &roles; manager
          </AttributeValue>
          <SubjectAttributeDesignator
            AttributeId = "&role;"
            DataType = "&xml; string"/>
        </SubjectMatch>
      </Subject>
    </Subjects>
  </Target>
  <PolicySetIdReference>PPS:manager:role</PolicySetIdReference>
</PolicySet>
```

Figure 4.7 Role <PolicySet> for a Manager

```xml
<ActionMatch MatchId = "&function; string-equal">
  <AttributeValue
    DataType = "&xml; string">
    update </AttributeValue>
  <ActionAttributeDesignator
    AttributeId = "&action; action-id"
    DataType = "&xml; string"/>
</ActionMatch>
  </Action>
</Actions>
</Target>
</Rule>
</Policy>
</PolicySet>
```
The <PolicySetIdReference> associates the Permission <PolicySet>, named "PPS:manager:role" with the Role <PolicySet>, named "RPS:manager:role".

```xml
<Policy xmlns="urn:oasis:names:tc:xacml:3.0:policy:schema:os"
    PolicyId = "Role:Assignment:Policy"
    RuleCombiningAlgId = "&rule-combine; permit-overrides">
    <Rule RuleId = "employee:role:requirements" Effect="Permit">
        <Target>
            <Subjects>
                <Subject>
                    <SubjectMatch MatchId = "&function; string-equal">
                        <AttributeValue
                            DataType = "&xml; string"> Ahmed </AttributeValue>
                        <SubjectAttributeDesignator
                            AttributeId = "&subject;subject-id"
                            DataType = "&xml; string"/>
                    </SubjectMatch>
                </Subject>
                <Subject>
                    <SubjectMatch MatchId = "&function; string-equal">
                        <AttributeValue
                            DataType = "&xml; string"> Mohammed </AttributeValue>
                        <SubjectAttributeDesignator
                            AttributeId = "&subject;subject-id"
                            DataType = "&xml; string"/>
                    </SubjectMatch>
                </Subject>
            </Subjects>
        </Target>
        <Resources>
            <Resource>
                <ResourceMatch MatchId = "&function; string-equal">
                    <AttributeValue
                        DataType="&xml; string"> &roles; employee </AttributeValue>
                    <ResourceAttributeDesignator
                        AttributeId = "&role;"
                        DataType = "&xml; string"/>
                </ResourceMatch>
            </Resource>
        </Resources>
    </Rule>
</Policy>
```

Figure 4.8 (Continued)
<Resources/>
<Actions>
  <Action>
    <ActionMatch MatchId="&function; string-equal">
      <AttributeValue
        DataType="&xml; string">
        &actions; enableRole
      </AttributeValue>
      <ActionAttributeDesignator
        AttributeId="&action; action-id"
        DataType="&xml; string"/>
    </ActionMatch>
  </Action>
</Actions>
</Target>
<Condition>
  <Apply FunctionId="&function; and">
    <Apply FunctionId="&function; time-greater-than-or-equal">
      <Apply FunctionId="&function; time-one-and-only">
        <EnvironmentAttributeDesignator
          AttributeId="&environment; current-time"
          DataType="&xml; time"/>
      </Apply>
      <AttributeValue
        DataType="&xml; time">
        9h
      </AttributeValue>
    </Apply>
    <Apply FunctionId="&function; time-less-than-or-equal">
      <Apply FunctionId="&function; time-one-and-only">
        <EnvironmentAttributeDesignator
          AttributeId="&environment; current-time"
          DataType="&xml; time"/>
      </Apply>
      <AttributeValue
        DataType="&xml; time">
        17h
      </AttributeValue>
    </Apply>
  </Apply>
</Condition>
</Rule>
</Policy>

Figure 4.8 Role Assignment <Policy>
In Figure 4.8, the <Rule> indicates that, Ahmed and Mohammed are allowed to have the "&roles; employee" role enabled between the hours of 9 am and 5 pm. Moreover, the attributes of the role are described as Resource attribute values in the Role Assignment <Policy>.

In the RBAC profile of XACML, the REA is the component that assigns role attributes and values to users during a user's session. The REA assigns users to roles. It uses the XACML Role Assignment <Policy> or <PolicySet> to specify whether a subject is permitted to be a member of a particular role attribute, and value enabled or not. The REA may maintain a list of all roles in an organization and when asked about the role that can be assigned to a user, it makes a request to the Role Assignment Policies for each candidate role. If a user is allowed to be a member of multiple roles, any Role <PolicySet> instance applying to any of those roles may be evaluated, and the permissions from the corresponding Permission <PolicySet> will be permitted.

The XACML policies, using the RBAC profile of XACML, can use a <Condition> to set constraints on the application of a particular permission. By using a <Condition>, the RBAC profile can limit the permissions associated with a certain role, according to the location of the user, and the suitable time of his request.

4.7 SUMMARY

This chapter depicts the standard XACML and the RBAC profile of XACML. For the standard XACML, it depicts the rule, policy, and policy set of the XACML policy language. Moreover, it presents the XACML request and response language.

For the RBAC profile of XACML, it depicts the RBAC briefly and explains the role, and the types of policy of the RBAC profile of XACML with examples.
CHAPTER 5

XML ACCESS CONTROL: MAPPING XACML POLICIES AND RULES INTO TEMPORAL RELATIONS

Although XACML is an integral policy description language, the structure of the existing XACML policy is very complex and hard. Therefore, it is necessary for the users to understand XACML well, in order to write all the securing policy specifications. On the other hand, the Query languages of a RDBMS are easy and simple to use by all the users. Moreover, SQL-like query languages overcome the difficulties of XACML, by storing the XACML policies and rules in relations. Therefore, it is easy for the users to use and understand the XACML policies and rules. Moreover, storing the data and rules in tables provides more flexibility. Hence, it is necessary to provide the table based storage features of web databases. This chapter proposes new algorithms for mapping the XACML policies and rules into relational rules, and storing them in the form of relational rules in temporal relations, to ease the access control to the XML documents. The results of the new algorithms are temporal relations, because the XACML polices may have a time-stamp in the <Environment> or in the <Condition>. Moreover, these algorithms are capable for mapping the policies of the RBAC profile of XACML that may have a time-stamp in the <Environment> or in the <Condition>. The implementation of the proposed algorithms demonstrates a significant access decision time.
5.1 INTRODUCTION

XML is the de facto standard for data representation and exchange over the web, and it is adopted in communication networks. XML documents over the World Wide Web include sensitive data, such as financial and scientific data. To protect these critical data, it is necessary to apply protection techniques on the content of the XML documents. Hence, only users who are authorized to access the XML documents are granted access to sensitive data (Patel & Atay 2011).

According to Abassi et al (2010), an access control algorithm for XML documents must provide expressiveness, modularity, interoperability, and efficiency. Expressiveness helps to write a wide range of security policy specifications effectively. Modularity provides flexibility for policies composition, and interoperability provides the ability for the access control policies to interact. Finally, efficiency is useful to determine whether an access to an element can be granted or denied by the security policy.

The XML documents can be stored in either native XML databases, relational databases, or a hybrid of both relational and XML databases. In the past, many techniques were proposed by various researchers to secure native XML databases. However, there are only a few techniques, which are proposed to control the access to the XML documents stored in relational databases (Tan et al 2001, Lee et al 2003, Luo et al 2007, and Patel & Atay 2011). Lee et al (2003), Luo et al (2007) proposed techniques for storing schema-less XML documents in relational databases. Tan et al (2001) proposed a new technique, which is not pure relational but enforces the access control rules from outside the relational engine, thus leading to a decrease in performance. Patel & Atay (2011) proposed an access control model, named XML to Relational Authorization Rule (X2RAC), for the storage of the XML documents in relations. Their model allows the security administrators to
specify the authorization rules in XML, and store them in relations. However, their technique maps only rules but not policies. Moreover, they included several issues in the mapping of an object parameter, and its condition to their relational entities, which caused some performance overhead.

XACML is the result of OASIS, which is useful to express and interchange access control policies and requests/responses effectively (Rissanen 2013). The XACML standard defines a declarative access control policy language, which has been implemented in XML. It also provides a processing model, which describes the techniques on how to evaluate the authorization requests according to the rules defined in the policies. This language offers the functionalities present in most of the security policy languages. Moreover, the structure of the XACML policy language is more complex, and difficult to learn and use for users. To overcome all these problems, this thesis proposes new algorithms that hide the complexity of the XACML policy definition in a RDBMS. These new algorithms help to store the XACML policies in temporal relations effectively. Hence, the proposed algorithms store the XACML policies and rules, in the form of rules in a relational database system, and hence, can control the access to the XML documents effectively.

The major advantages of this proposed work are:

1. It relieves the users from the effort of learning and understanding the XACML policies and rules; hence, it leads to reduction in the users' time and effort.

2. It controls the access to the XML documents stored in either native or relational databases, using the XACML policies.

3. It applies the constraints of the rules, and obligations and provides the response to an access request effectively.
5.2 MAPPING XACML POLICIES AND RULES INTO TEMPORAL RELATIONS

XACML is one of the most efficient languages used for describing access control policies to web documents. However, it has a complicated representation structure, and is not simple to manipulate for non-expert users. Hence, this thesis proposes new algorithms to overcome this complexity, and store the XACML policies and rules, in the form of relational rules. Moreover, triggers are created for these temporal relations, to provide integrity and security. This section introduces the proposed work that maps the XACML policies and rules into temporal relations.

5.2.1 Proposed Mapping Algorithm

The procedure of the proposed mapping algorithm, which facilitates the mapping of the XACML policies into relational rules, and storing them in the form of relational rules in temporal relations, is presented in Figure 5.1:

<table>
<thead>
<tr>
<th>Input</th>
<th>An XACML rule, policy, or policy set document.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output</td>
<td>Authorization relations and corresponding triggers.</td>
</tr>
<tr>
<td>BEGIN:</td>
<td></td>
</tr>
<tr>
<td>Step 1: Parse the XACML document.</td>
<td></td>
</tr>
<tr>
<td>Step 2: IF a &lt;PolicySet&gt; is the root</td>
<td></td>
</tr>
<tr>
<td>Call PolicySet_as_root ().</td>
<td></td>
</tr>
<tr>
<td>Step 3: ELSE IF a &lt;Policy&gt; is the root</td>
<td></td>
</tr>
<tr>
<td>Call Policy_as_root ().</td>
<td></td>
</tr>
<tr>
<td>Step 4: ELSE</td>
<td></td>
</tr>
<tr>
<td>Call Rule_as_root ().</td>
<td></td>
</tr>
<tr>
<td>END IF.</td>
<td></td>
</tr>
<tr>
<td>END.</td>
<td></td>
</tr>
</tbody>
</table>

Figure 5.1 Procedure of the Proposed Mapping Algorithm
This mapping algorithm begins by parsing the XACML document, and checking which the root of the document is. If the `<PolicySet>` is the root, it calls the method `PolicySet_as_root()`. Else, if the `<Policy>` is the root, it calls the method `Policy_as_root()`. Else, it calls the method `Rule_as_root()`. The three methods are presented as follows:

### 5.2.1.1 Policy Set is the Root

When the root of the XACML document is a `<PolicySet>`, the mapping algorithm calls the method `PolicySet_as_root()`. The steps of the algorithm of the method `PolicySet_as_root()` are as follows:

**PolicySet_as_root()**:  

**BEGIN:**

**Step 1:** Create a PolicySet table:

\[
\text{PolicySet (PolicySetID, PCAlgId, Description)}.
\]

**Step 2:** IF the `<Target>` is non-empty, create a PolicySetTarget table:

\[
\text{PSTarget (PSTargetID, PolicySetID)} \text{ and call Check_target_components (PSTargetID)}.
\]

END IF.

**Step 3:** IF the `<PolicySet>` has `<Policy>` elements:

a. Create a Policy table:

\[
\text{Policy (PolicyID, RCAlgId, Description, PolicySetID)}.
\]

b. FOR each `<Policy>` do:

i. IF the `<Target>` is non-empty, create a PolicyTarget table:

\[
\text{PTarget (PTargetID, PolicyID)} \text{ and call Check_target_components (PTargetID)}.
\]
ii. END IF.

iii. Call Create_rule_table (PolicyID).

iv. IF the <Policy> has an <ObligationExpressions>, call Create_policy_obligation_table (PolicyID).

v. END IF.
c. END FOR.

END IF.

END.

The steps of the algorithm of the method Create_rule_table (PolicyID_fk) are as follows:

Create_rule_table (PolicyID_fk):

BEGIN:

Step 1: Create a Rule table:

Rule (RuleID, Effect, Description, PolicyID_fk).

Step 2: FOR each <Rule> do:

a. IF a <Target> exists and is non-empty, create a RuleTarget table: RTarget (RTargetID, RuleID) and call Check_target_components (RTargetID).

b. END IF.

c. IF the <Rule> has a <Condition>, call Create_rule_condition_table (RuleID).

d. END IF.

e. IF the <Rule> has an <ObligationExpressions>, call
The steps of the algorithm of the method Create_rule_obligation_table (RuleID) are as follows:

Create_rule_obligation_table (RuleID).

f. END IF.

END FOR.

END.

The steps of the algorithm of the method Create_rule_condition_table (RuleID_fk) are as follows:

Create_rule_condition_table (RuleID_fk):

BEGIN:

Step 1: Create a Condition table: Condition (CondID, RuleID_fk).

Step 2: Create a CondApply table:

CondApply (AppID, CondID, FuncID, AppID_fk, CondID_fk).

Step 3: FOR each <Apply> do:

a. IF it has an <AttributeValue>, create an ApplyAttVal table:

ApplyAttVal (ID, AttDT, AttVal, AppID, CondID).

b. ELSE IF it has an <AttributeDesignator>, create an ApplyAttDes table: ApplyAttDes (ID, AttDesID, AttDesDT, AttDesCat, AppID, CondID).

c. ELSE create an ApplyAttSel table: ApplyAttSel (ID, Path, AttSelDT, AttSelCat, AppID, CondID).

d. END IF.

END FOR.

END.

The steps of the algorithm of the method Create_rule_obligation_table (RuleID_fk) are as follows:
Create_rule_obligation_table (RuleID_fk):

BEGIN:

Step 1: Create a ROblExp table: ROblExp (OID, OblID, FulfillOn, RuleID_fk).

Step 2: FOR each <ObligationExpression> do:

a. FOR each <AttributeAssignmentExpression> do:

i. IF it has an <AttributeDesignator>, create a ROblAttDes table: ROblAttDes (ID, AttDesID, AttDesDT, AttDesCat, OID).

ii. ELSE IF it has an <AttributeSelector>, create a ROblAttSel table: ROblAttSel (ID, Path, AttSelDT, AttSelCat, OID).

iii. ELSE create a ROblAttVal table: ROblAttVal (ID, AttDT, AttVal, OID).

iv. END IF.

b. END FOR.

END FOR.

END.

The steps of the algorithm of the method Create_policy_obligation_table (PolicyID_fk) are as follows:

Create_policy_obligation_table (PolicyID_fk):

BEGIN:

Step 1: Create a PObIExp table: PObIExp (OID, OblID, FulfillOn, PolicyID_fk).
**Step 2:** FOR each `<ObligationExpression>` do:

a. FOR each `<AttributeAssignmentExpression>` do:
   
i. IF it has an `<AttributeDesignator>`, create a `POblAttDes` table: `POblAttDes (ID, AttDesID, AttDesDT, AttDesCat, OID)`.
   
ii. ELSE IF it has an `<AttributeSelector>`, create a `POblAttSel` table: `POblAttSel (ID, Path, AttSelDT, AttSelCat, OID)`.
   
iii. ELSE create a `POblAttVal` table: `POblAttVal (ID, AttDT, AttVal, OID)`.
   
iv. END IF.

b. END FOR.

END FOR.

END.

As stated above, the steps of the algorithm of the method `PolicySet_as_root()` are executed as follows:

1. **Step 1** creates a table, called a PolicySet table:

   \[
   \text{PolicySet (PolicySetID, PCAlgId, Description).}
   \]

2. **Step 2** checks whether the `<Target>` of the `<PolicySet>` is empty or not. If it is non-empty, it creates a PSTarget table:

   \[
   \text{PSTarget (PSTargetID, PolicySetID) and calls the method Check_target_components (PSTargetID).}
   \]
3. In step 3, if the <PolicySet> has <Policy> elements, step 3.a creates a Policy table:

   **Policy** (**PolicyID**, RCAAlgId, Description, PolicySetID).

4. Step 3.b loops over all the <Policy> elements and for each one, it calls the method `Create_rule_table (PolicyID)` and checks whether the <Policy> has a <Target> and an <ObligationExpressions> or not.

5. If the <Policy> has a non-empty <Target>, step 3.b.i creates a PTarget table: **PTarget** (**PTargetID**, PolicyID) and calls the method `Check_target_components (PTargetID)`.

6. For <Rule> elements, step 3.b.iii calls the method `Create_rule_table (PolicyID)` that creates a Rule table: **Rule** (**RuleID**, Effect, Description, PolicyID) and loops over all the <Rule> elements, to check whether each <Rule> has a <Target>, a <Condition>, and an <ObligationExpressions> or not.

7. If the <Rule> has a non-empty <Target>, the algorithm of the method `Create_rule_table (PolicyID)` creates an RTarget table: **RTarget** (**RTargetID**, RuleID) and calls the method `Check_target_components (RTargetID)`.

8. If the <Rule> has a <Condition>, the algorithm of the method `Create_rule_table (PolicyID)` calls the method `Create_rule_condition_table (RuleID)`. The algorithm of the method `Create_rule_condition_table (RuleID)` creates a Condition table: **Condition** (**CondID**, RuleID) and a CondApply table: **CondApply** (**AppID**, **CondID**, FuncID, AppID_fk, CondID_fk), to store the function of each <Apply>
and the functions of its children and descendants. Moreover, the algorithm of the method Create_rule_condition_table (RuleID) creates:

- An ApplyAttVal table: **ApplyAttVal (ID, AttDT, AttVal, AppID, CondID),**

- An ApplyAttDes table: **ApplyAttDes (ID, AttDesID, AttDesDT, AttDesCat, AppID, CondID),**

- Or an ApplyAttSel table: **ApplyAttSel (ID, Path, AttSelDT, AttSelCat, AppID, CondID),** to store the arguments of each <Apply>.

9. If the <Rule> has an <ObligationExpressions>, the algorithm of the method Create_rule_table (PolicyID) calls the method Create_rule_obligation_table (RuleID). The algorithm of the method Create_rule_obligation_table (RuleID) creates a ROblExp table: **ROblExp (OID, OblID, FulfillOn, RuleID)** and loops over all the <ObligationExpression> and <AttributeAssignmentExpression> elements, to create either:

- A ROblAttDes table: **ROblAttDes (ID, AttDesID, AttDesDT, AttDesCat, OID),**

- A ROblAttSel table: **ROblAttSel (ID, Path, AttSelDT, AttSelCat, OID),**

- Or a ROblAttVal table: **ROblAttVal (ID, AttDT, AttVal, OID),** to store the attribute values that describe the action specified by the rule's obligation.

10. If the <Policy> has an <ObligationExpressions>, **step 3.b.iv** calls the method Create_policy_obligation_table (PolicyID).
The algorithm of the method Create_policy_obligation_table (PolicyID) creates a POblExp table: **POblExp (OID, OblID, FulfillOn, PolicyID)** and loops over all the <ObligationExpression> and <AttributeAssignmentExpression> elements, to create either:

- A POblAttDes table: **POblAttDes (ID, AttDesID, AttDesDT, AttDesCat, OID),**
- A POblAttSel table: **POblAttSel (ID, Path, AttSelDT, AttSelCat, OID),**
- Or a POblAttVal table: **POblAttVal (ID, AttDT, AttVal, OID),** to store the attribute values that describe the action specified by policy's obligation.

The algorithm for checking the target components, called Check_target_components (TargetID_fk) is presented as follows:

**Check_target_components (TargetID_fk):**

BEGIN:

**Step 1:** IF a <Subjects> exists and is non-empty, create a Subjects table:

**Subjects (ID, SubMatchID, AttVal, AttDT, AttDesID,**

AttDesDT, AttDesCat, TargetID_fk).

END IF.

**Step 2:** IF a <Resources> exists and is non-empty, create a Resources table: **Resources (ID, ResMatchID, AttVal, AttDT, AttDesID,**

AttDesDT, AttDesCat, TargetID_fk).

END IF.
**Step 3:** IF an `<Actions>` exists and is non-empty, create an Actions table:

\[ \text{Actions (ID, ActionMatchID, AttVal, AttDT, AttDesID,} \]
\[ \text{AttDesDT, AttDesCat, TargetID_fk).} \]

END IF.

**Step 4:** IF an `<Environments>` exists and is non-empty, create an

\[ \text{Environments (ID, EnvMatchID, AttVal,} \]
\[ \text{AttDT, AttDesID, AttDesDT, AttDesCat, TargetID_fk).} \]

END IF.

END.

The steps of the algorithm of the method `Check_target_components(TargetID_fk)` are executed as follows:

1. **Step 1** checks whether a `<Subjects>` exists and is non-empty or not. If the `<Subjects>` exists and is non-empty, it creates a Subjects table: \[ \text{Subjects (ID, SubMatchID, AttVal, AttDT,} \]
\[ \text{AttDesID, AttDesDT, AttDesCat, TargetID_fk).} \]

2. **Step 2** checks whether a `<Resources>` exists and is non-empty or not. If the `<Resources>` exists and is non-empty, it creates a Resources table: \[ \text{Resources (ID, ResMatchID, AttVal, AttDT,} \]
\[ \text{AttDesID, AttDesDT, AttDesCat, TargetID_fk).} \]

3. **Step 3** checks whether an `<Actions>` exists and is non-empty or not. If the `<Actions>` exists and is non-empty, it creates an Actions table: \[ \text{Actions (ID, ActionMatchID, AttVal, AttDT,} \]
\[ \text{AttDesID, AttDesDT, AttDesCat, TargetID_fk).} \]

4. **Step 4** checks whether an `<Environments>` exists and is non-empty or not. If the `<Environments>` exists and is non-empty, it
creates an Environments table: \textbf{Environments} (\textbf{ID}, EnvMatchID, AttVal, AttDT, AttDesID, AttDesDT, AttDesCat, TargetID\_fk).

5.2.1.2 Policy is the Root

When a \texttt{<Policy>} is the root, the mapping algorithm invokes the method \texttt{Policy\_as\_root ()}. The steps of the algorithm of the method \texttt{Policy\_as\_root ()} are as follows:

\texttt{Policy\_as\_root ():}

\texttt{BEGIN:}

\texttt{Step 1: Create a Policy table: Policy (PolicyID, RCAlgId, Description).}

\texttt{Step 2: IF the \texttt{<Target>} is non-empty, create a PolicyTarget table:}

\texttt{PTarget (PTargetID, PolicyID) and call Check\_target\_components (PTargetID).}

\texttt{END IF.}

\texttt{Step 3: Call Create\_rule\_table (PolicyID).}

\texttt{Step 4: IF the \texttt{<Policy>} has an \texttt{<ObligationExpressions>}, call Create\_policy\_obligation\_table (PolicyID).}

\texttt{END IF.}

\texttt{END.}

The steps of the algorithm of the method \texttt{Policy\_as\_root ()} are executed as follows:

1. \texttt{Step 1} creates a Policy table: \texttt{Policy (PolicyID, RCAlgId, Description).}
2. **Step 2** checks whether the `<Policy>` has a non-empty `<Target>` or not. If the `<Target>` is non-empty, it creates a PTarget table and calls the method Check_target_components (PTargetID).

3. **Step 3** calls the method Create_rule_table (PolicyID). The algorithm of the method Create_rule_table (PolicyID) creates a Rule table and loops over all the `<Rule>` elements, to checks whether each `<Rule>` has a `<Target>`, a `<Condition>`, and an `<ObligationExpressions>` or not.

4. If the `<Policy>` has an `<ObligationExpressions>`, **step 4** calls the method Create_policy_obligation_table (PolicyID). The algorithm of the method Create_policy_obligation_table (PolicyID) creates a POblExp table and loops over all the `<ObligationExpression>` and over all the `<AttributeAssignmentExpression>` elements, to create for each `<AttributeAssignmentExpression>` either a POblAttDes, POblAttSel, or POblAttVal table, to store the attribute values that describe the action specified by the policy's obligation.

5.2.1.3 **Rule is the Root**

Finally, when a `<Rule>` is the root of the XACML document, the mapping algorithm calls the method Rule_as_root (). The steps of the algorithm of the method Rule_as_root () are as follows:

**Rule_as_root ():**

**BEGIN:**

**Step 1:** Create a Rule table: Rule (RuleID, Effect, Description).

**Step 2:** IF a `<Target>` exists and is non-empty, create a RuleTarget table:
**RTarget (RTargetID, RuleID)** and call Check_target_components (RTargetID).

END IF.

**Step 3:** IF the <Rule> has a <Condition>, call

Create_rule_condition_table (RuleID).

END IF.

**Step 4:** IF the <Rule> has an <ObligationExpressions>, call

Create_rule_obligation_table (RuleID).

END IF.

END.

The above code of the method runs as follows:

1. **Step 1** creates a Rule table: **Rule (RuleID, Effect, Description).**

2. If the <Rule> has a non-empty <Target>, **step 2** creates an RTarget table and calls the method Check_target_components (RTargetID).

3. If the <Rule> has a <Condition>, **step 3** calls the method Create_rule_condition_table (RuleID). The algorithm of the method Create_rule_condition_table (RuleID) creates a Condition table and a CondApply table, to store the function of each <Apply> and the functions of its children and descendants. Moreover, the algorithm of the method Create_rule_condition_table (RuleID) creates tables, namely, an ApplyAttVal, ApplyAttDes, or ApplyAttSel, to store the arguments of each <Apply>. 
4. If the <Rule> has an <ObligationExpressions>, step 4 calls the method Create_rule_obligation_table (RuleID). The algorithm of the method Create_rule_obligation_table (RuleID) creates a ROblExp table and loops over all the <ObligationExpression> and over all the <AttributeAssignmentExpression> elements, to create for each <AttributeAssignmentExpression> either a ROblAttDes, ROblAttSel, or ROblAttVal table, to store the attribute values that describe the action specified by the rule's obligation.

In addition to the mapping of the rule, policy, or policy set documents, the proposed mapping algorithm also maps the request documents into relations, using the same idea of the method Check_target_components (arg).

5.3 IMPLEMENTATION AND RESULTS

The implementation of the proposed mapping algorithm has been carried out in Java platform, using MS Access as a database. The MS Access tables are used for storing the XACML policies and rules in the forms of relational rules, and triggers have been created for these tables, to provide integrity and security. The performance evaluation of the mapping algorithm has been done on a Toshiba laptop running Windows 7 with Pentium (R) Dual-Core CPU 2.20 GHz and 3GB RAM. Before using the application, the XACML document has been enhanced as follows:

1. If the <PolicySet>, <Policy>, or <Rule> has a <Target>, change the <Target> to be <PSTarget>, <PTarget>, and <RTarget> respectively, to differentiate among them in parsing.
2. If the <Policy> or <Rule> has an <ObligationExpressions>, change it to <PObligationExpressions> and <RObligationExpressions> respectively, to distinguish between them in parsing.

3. Remove the empty elements.

5.3.1 Creating Authorization Relations

The following example explains the applying of the proposed mapping algorithm to the XACML Policy Set document in Figure 5.2, and the Request document in Figure 5.3:

```xml
<PolicySet PolicySetId = "pls-0001" PolicyCombiningAlgId = "urn:oasis:names:tc:xacml:3.0:policy-combining-algorithm:deny-overrides">
  <Description>
    It is the description of the policy set
  </Description>
</PolicySet>

<PSTarget>
  <Subjects>
    <Subject>
      <SubjectMatch MatchId = "urn:oasis:names:tc:xacml:1.0:function:string-equal">
        <AttributeValue DataType = "http://www.w3.org/2001/XMLSchema#string">
          Omar
        </AttributeValue>
      </SubjectMatch>
    </Subject>
  </Subjects>
</PSTarget>
```

Figure 5.2 (Continued)
<Policy PolicyId = "pol-0001" RuleCombiningAlgId = "urn:oasis:names:tc:xacml:3.0: rule-combining- algorithm:deny-overrides">
  <Description>
    It is the description of the policy
  </Description>
  <Rule RuleId = "rul-0001" Effect = "Permit">
    <Condition>
      <Apply FunctionId = "urn:oasis:names:tc:xacml:1.0:function:string-equal">
        <Apply FunctionId = "urn:oasis:names:tc:xacml:1.0:function:string-one-and-only">
          <AttributeDesignator
            AttributeId = "urn:oasis:names:tc:xacml:1.0:subject:subject-id"
            DataType = "http://www.w3.org/2001/XMLSchema#string"/>
        </Apply>
        <AttributeValue
          DataType = "http://www.w3.org/2001/XMLSchema#string">
          Omar
        </AttributeValue>
      </Apply>
    </Condition>
  </Rule>
  <Rule RuleId = "rul-0002" Effect = "Permit">
    <Condition>
      <Apply FunctionId = "urn:oasis:names:tc:xacml:1.0:function:string-equal">
        <Apply FunctionId = "urn:oasis:names:tc:xacml:1.0:function:string-one-and-only">
          <AttributeDesignator
            AttributeId = "urn:oasis:names:tc:xacml:1.0:subject:subject-id"
            DataType = "http://www.w3.org/2001/XMLSchema#string"/>
        </Apply>
        <AttributeValue
          DataType = "http://www.w3.org/2001/XMLSchema#string">
          Ahmed
        </AttributeValue>
      </Apply>
    </Condition>
  </Rule>
</Policy>

Figure 5.2 (Continued)
Figure 5.2 Example of an XACML Policy Set Document
<Request>
  <Subject>
    <Attributes>
      <Attribute
        AttributeId = "urn:oasis:names:tc:xacml:1.0:subject:subject-id">
        <AttributeValue
          DataType = "http://www.w3.org/2001/XMLSchema#string">
          Omar
        </AttributeValue>
      </Attribute>
    </Attributes>
  </Subject>
  <Resource>
    <Attributes>
      <Attribute
        AttributeId = "urn:oasis:names:tc:xacml:1.0:resource:resource-id">
        <AttributeValue
          DataType = "http://www.w3.org/2001/XMLSchema#string">
          http://Hospital.com/record/patient/Zizo
        </AttributeValue>
      </Attribute>
    </Attributes>
  </Resource>
  <Action>
    <Attributes>
      <Attribute
        AttributeId = "urn:oasis:names:tc:xacml:1.0:action:action-id">
        <AttributeValue
          DataType = "http://www.w3.org/2001/XMLSchema#string">
          Read
        </AttributeValue>
      </Attribute>
    </Attributes>
  </Action>
</Request>

Figure 5.3 (Continued)
Figure 5.3 Example of an XACML Request Document

When the application of the mapping algorithm is running, a main form appears that allows the user to select the Policy Set document or the Request document. Hence, when the selection is the Policy Set document as in Figure 5.2 or the Request document as in Figure 5.3, the application shows a form with the corresponding relations of the Policy Set document or the Request document, according to the proposed mapping algorithm. The application executes as follows:

1. When the selection is the Policy Set document in Figure 5.2, the application executes step 1 of the mapping algorithm that
parses the XACML Policy Set document and find that, the root is a `<PolicySet>`.

2. Step 2 of the mapping algorithm calls the method PolicySet_as_root() that creates the following authorization relations:

   **PolicySet** (PolicySetID, PCAlgId, Description).

   **PSTarget** (PSTargetID, PolicySetID).

   **PSTargetSubjects** (ID, SubMatchID, AttVal, AttDT, AttDesID, AttDesDT, AttDesCat, PSTargetID).

   **Policy** (PolicyID, RCAlgId, Description, PolicySetID).

   **Rule** (RuleID, Effect, Description, PolicyID).

   **Condition** (CondID, RuleID).

   **CondApply** (AppID, CondID, FuncID, AppID_fk, CondID_fk).

   **ApplyAttVal** (ID, AttDT, AttVal, AppID, CondID).

   **ApplyAttDes** (ID, AttDesID, AttDesDT, AttDesCat, AppID, CondID).

   **POblExp** (OID, OblID, FulfillOn, PolicyID).

   **POblAttDes** (ID, AttDesID, AttDesDT, AttDesCat, OID).

   **POblAttSel** (ID, Path, AttSelDT, AttSelCat, OID).

   **POblAttVal** (ID, AttDT, AttVal, OID).

3. When the selection is the Request document in Figure 5.3, the application creates the following relations:

   **Request** (ReqID).

   **ReqSubject** (SubID, ReqID).

   **SubjectAtt** (ID, Cat, AttID, AttVal, AttDT, SubID).
\textbf{ReqResource} (\textbf{ResID}, ReqID).


\textbf{ReqAction} (\textbf{ActionID}, ReqID).

\textbf{ActionAtt} (\textbf{ID}, Cat, AttID, AttVal, AttDT, ActionID).

\textbf{ReqEnv} (\textbf{EnvID}, ReqID).

\textbf{EnvAtt} (\textbf{ID}, Cat, AttID, AttVal, AttDT, EnvID).

Therefore, by using the authorization relations, the access control to the XML documents will be easier and more effective, than while, using the complex structure of the XACML policies.

\textbf{5.3.2 Querying on Authorization Relations}

Now, the authorization relations can be used to provide the conditions results, obligations, and response to the request as follows:

Since the application created a Condition relation, the following condition query is executed to get the results of the conditions that may be true or false:

\begin{verbatim}
SELECT RuleID, CondID, IIf ([SubjectAtt.AttVal] = [ApplyAttVal.AttVal], True, False) AS CondRes
FROM CondApply, Condition, ApplyAttDes, ApplyAttVal, SubjectAtt
\end{verbatim}
This query retrieves the name value of the subject and compares it with the two name values stated in the two conditions. If the name value of the subject equals any name value of the two name values stated in the two conditions, the query returns True. Otherwise, it returns False. If the result of the condition query is False, the corresponding rule is NotApplicable. Otherwise, the corresponding rule is applicable. The results of the condition query are shown in Table 5.1:

<table>
<thead>
<tr>
<th>RuleID</th>
<th>CondID</th>
<th>CondRes</th>
</tr>
</thead>
<tbody>
<tr>
<td>rul-0001</td>
<td>1</td>
<td>True</td>
</tr>
<tr>
<td>rul-0002</td>
<td>2</td>
<td>False</td>
</tr>
</tbody>
</table>

Table 5.1 states that, the result of the condition of the first rule is True; thus it is applicable, and the result of the condition of the second rule is False; thus it is NotApplicable. Since the first rule is applicable, the response query is executed to get the Effect of the first rule:

```sql
FROM Rule, Policy, PolicySet, ConditionQuery, PSTarget, PSTargetSubjects, SubjectAtt, Request, ReqSubject, ReqAction, ActionAtt
```

WHERE

The using of the RTarget, PTarget, or PSTarget relation in the response query is based on the following:

- IF the RTarget relation is non-empty, use it in the response query.
- ELSE IF the PTarget relation is non-empty, use it in the response query.
- ELSE use the PSTarget relation in the response query.
END IF.

Therefore, the PSTarget is used in the response query and the join is done between the PSTarget and SubjectAtt relations, because the RTarget and PTarget relations are empty.

The wizard creates the above response query in an easy way, but the SQL is used, instead of using the figures of the wizard. The rules' effects are shown in Table 5.2:

**Table 5.2 Rules' Effects**

<table>
<thead>
<tr>
<th>RuleID</th>
<th>FEffec</th>
<th>CondRes</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>rul-0001</td>
<td>Permit</td>
<td>True</td>
<td>read</td>
</tr>
<tr>
<td>Rul-0002</td>
<td>NotApplicable</td>
<td>False</td>
<td>read</td>
</tr>
</tbody>
</table>

Table 5.2 states that, the evaluation of the first rule is Permit and the evaluation of the second rule is NotApplicable. Hence, according to the
rule-combining algorithm, named Deny overrides algorithm, the policy evaluates to Permit, and according to the policy-combining algorithm, named Deny overrides algorithm, the policy set evaluates to Permit.

Since the response permits the access request to read the resource, the obligation must be performed. To know the obligation and perform it, the following obligation query is executed:

```
SELECT OblID, FulfillOn, Path, SubjectAtt.AttVal AS Subject,
       POblAttVal.AttVal AS Value
FROM POblExp, POblAttDes, POblAttSel, POblAttVal, SubjectAtt
WHERE POblExp.OID = POblAttSel.OID and POblExp.OID = POblAttDes.OID and POblExp.OID = POblAttVal.OID and POblAttDes.AttDesDT = SubjectAtt.AttDT and POblAttDes.AttDesID = SubjectAtt.AttID;
```

From the above obligation query, the result is shown in Table 5.3:

<table>
<thead>
<tr>
<th>OblID</th>
<th>FulfillOn</th>
<th>Path</th>
<th>Subject</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>email</td>
<td>Permit</td>
<td>Zizo/Contact/email</td>
<td>Omar</td>
<td>Your record is accessed</td>
</tr>
</tbody>
</table>

In Table 5.3, the obligation is sending an email to the subject Omar. The email is sent to Omar when the response's effect is Permit. The email address of the subject Omar is an element, called email, contained in the resource, called Zizo element, and its path is Zizo/Contact/email. If the resource is stored in native XML databases, the email address of the subject is retrieved by an XPath query. If the resource is stored in relational databases,
the XPath query is translated into an SQL query on the resulting relations of the resource. The body of the email is "Your record is accessed".

5.4 SUMMARY

In this chapter, new algorithms for mapping the XACML policies and rules into relational rules, and storing them in the form of rules in temporal relations, are proposed. The resulting temporal relations control the access to the XML documents effectively. These proposed algorithms relieve the users from the complexity of the XACML structure, and allow them to efficiently control the access to the XML documents, leading to reduction in their time and effort. From the experimental results obtained from this work, it is clearly proved that, these proposed algorithms map the XACML policies and rules effectively.
CHAPTER 6

A TIME-STAMP BASED ALGORITHM FOR Decrypting THE ENCRYPTED XML DOCUMENTS PARTIALLY

When a SOAP web service receives an encrypted XML document, it decrypts the key and uses it to decrypt all the parts of the document containing the encrypted data. Finally, the data are parsed with an XML parser, and the whole document is forwarded to the next module. In this case, the XML security processing module typically does not know which parts of the decrypted data are later processed by the business logic, due to the absence of a time-stamp during the decryption process. Thus, it could also happen that, the encrypted data are decrypted, parsed, and not processed further for security analysis. This chapter introduces a new time-stamp based algorithm, for decrypting the encrypted XML documents partially. This proposed algorithm utilizes a new temporal model for XML representation, based on a combination of valid time and transaction time, through the tuple time-stamping. In addition to the time-stamp, one attribute of the key components of the encrypted elements is selected to distinguish among them. The time-stamp and the selected attribute guide the receiver to decrypt the required parts of the encrypted XML documents, instead of decrypting all the parts of them. Moreover, they can be used to retrieve the parts that must be decrypted, to contribute to the query results on the encrypted XML documents. The proposed algorithm not only reduces the decryption time, but also protects the security of the data.
6.1 INTRODUCTION

Although XML encryption achieves the confidentiality of the transmitted data, the receiver must decrypt the entire document to get the required data, because there is no indication that the receiver can use it to determine the required data. In this chapter, a new time-stamp based algorithm for decrypting the encrypted XML documents partially is proposed. This proposed algorithm utilizes a new temporal model for XML representation, based on a combination of valid time and transaction time as the tuple time-stamping, and one selected attribute of the key components of the encrypted elements to distinguish among them. The time-stamp and the selected attribute help the receiver to get the required encrypted data of the encrypted XML documents to be decrypted. Hence, unnecessary decryption of the other parts of the encrypted data can be prevented. Moreover, the receiver can use the time-stamp and the selected attribute to decrypt the parts needed to answer queries on the encrypted XML documents. The proposed algorithm not only reduces the decryption time, but also protects the security of the data.

6.2 TIME REPRESENTATION IN XML DATABASES

Wang & Zaniolo (2004) represented the valid time and the transaction time in a hierarchical XML view (BH-document), as depicted in Figure 6.1. Each element in the BH-document is time-stamped (date-time, date, or time) using four XML attributes vstart, vend, tstart, and tend, which are called attributes time-stamp. The attributes vstart and vend represent the inclusive valid time interval, and tstart and tend represent the inclusive transaction time interval. The format of the time-stamp may be the date-time, time, or date, because the attribute value in XML is a string value. The value of tend can be assigned to UC (Until Changed), and vend can be assigned to now, to denote the ever-increasing current date. The parent element (e.g., employee Omar in Figure 6.1) always has a longer or equal lifespan than its children. Hence, the valid and transaction time intervals of a parent node always cover those of its child nodes.
Figure 6.1 Example of the Time Representation in XML Databases
The XML time representation in Figure 6.1 is called attribute time-stamping. In the proposed algorithm, the time representation is changed to tuple representation, instead of attribute representation; i.e., using the timestamp attributes only in the parent element, instead of using them in every element as in Wang & Zaniolo (2004), and a key attribute is added to each element. Moreover, the current and history data are separated in two XML documents. The proposed representation is depicted as in Figures 6.2 and 6.3.

```xml
<employees>
  <employee Id = 1 vstart = "01-01-2012" vend = "now"
            tstart = "01-01-2012"  tend = "UC">
    <name> Omar </name>
  </employee>
  <employee Id = 1 vstart = "01-07-2012" vend = "now"
            tstart = "01-06-2012"  tend = "UC">
    <title> manager </title>
    <salary> 9000 </salary>
  </employee>
</employees>
```

**Figure 6.2 Example of the Tuple XML Representation of the Current Data**

```xml
<employees>
  <employee Id = 1 vstart = "01-01-2012" vend = "31-06-2012"
            tstart = "01-01-2012"  tend = "31-05-2012">
    <title> employee </title>
    <salary> 5000 </salary>
  </employee>
</employees>
```

**Figure 6.3 Example of the Tuple XML Representation of the History Data**
When a new employee is inserted, the new employee element with its children element is appended in the current BH-document; the vstart is set to the valid starting time-stamp, the vend is set to now, the tstart is set to the transaction starting time-stamp, and the tend is set to UC. When an employee is deleted, his details are moved to the history BH-document and the vend is set to the end time of the deletion, and the tend is set to the last updating time of the vend.

### 6.3 PROPOSED TIME-STAMP BASED ALGORITHM

In this section, a new time-stamp based algorithm for decrypting the encrypted XML documents partially is proposed. This proposed algorithm utilizes a new temporal model for XML representation, based on a combination of valid time and transaction time as the tuple time-stamp. In addition to the time-stamp, one selected attribute of the key components of the encrypted elements is used to distinguish among the encrypted elements, to decrypt the required parts, instead of decrypting all the parts of the encrypted XML documents. The proposed algorithm is depicted in Figure 6.4:

**Input**: An XML document.

**Output**: Encrypted Current and History documents.

**BEGIN:**

**Step 1**: Choose a time-stamp format (date-time, time, or date) that is suitable for the organization, and select the attribute that can distinguish among the elements.

**Step 2**: Add the selected attribute as an attribute of each XML element.

**Step 3**: Apply the tuple XML time representation, and create a current BH-document and a history BH-document.

**Step 4**: Encrypt the current and history XML documents and send them to the receiver.

**Step 5**: The receiver uses the chosen time-stamp and the selected attribute, to make queries using the XQuery language on the encrypted current and encrypted history XML documents, to get the results that should be decrypted.

**END.**

*Figure 6.4 Proposed Time-Stamp Based Algorithm*
The proposed algorithm will be explained by the following example:

**Example**: Let us have the following XML document that represents employees:

```xml
<employees>
  <employee>
    <name> Omar </name>
    <title> manager </title>
    <salary> 5000 </salary>
  </employee>
  <employee>
    <name> Ahmed </name>
    <title> employee </title>
    <salary> 3000 </salary>
  </employee>
  <employee>
    <name> zizo </name>
    <title> manager </title>
    <salary> 4000 </salary>
  </employee>
</employees>
```

![Figure 6.5 Employees' XML Document](image)

After applying the XML encryption, the employees' XML document will be as follows:
Figure 6.6 Encrypted Employees' XML Document
In the encrypted employees' XML document that is depicted in Figure 6.6, the receiver cannot determine which parts must be decrypted. Therefore, the entire encrypted employees' XML document must be decrypted to get the required parts. To solve this problem, the proposed algorithm is used as follows: First, choosing the time-stamp format (as the date in our example) and the rank attribute that can distinguish among the elements. Second, adding the selected rank attribute as an attribute of each XML element. Third, applying the tuple XML time representation of the time-stamp (valid time and transaction time) as shown in section 6.2, by using four XML attributes only in the parent element, instead of using them in every element as in Wang & Zaniolo (2004) and creating a current BH-document and a history BH-document. Therefore, the current and history BH-documents are depicted as in Figures 6.7 and 6.8 respectively:

```xml
<employees>
  <employee rank = 1 vstart = "01-06-2012" vend = "now"
             tstart = "01-06-2012" tend = "UC">
    <name> Omar </name>
    <title> manager </title>
    <salary> 5000 </salary>
  </employee>
  <employee rank = 2 vstart = "01-01-2012" vend = "now"
             tstart = "02-01-2012" tend = "UC">
    <name> Ahmed </name>
    <title> employee </title>
    <salary> 3000 </salary>
  </employee>
</employees>
```

**Figure 6.7 Current BH-Document**
<employees>
  <employee rank = 1 vstart = "01-01-2012" vend = "31-12-2012"
tstart = "01-01-2012" tend = "01-01-2013">
    <name> zizo </name>
    <title> manager </title>
    <salary> 4000 </salary>
  </employee>
</employees>

Figure 6.8 History BH-Document

The valid time represents the time of designation in reality, and the rank attribute represents the rank of the employees (i.e., rank =1 means a manager and rank =2 means an employee).

Fourth, the flexibility of XML encryption allows encrypting only the content of each XML element in the current and history BH-documents, and sending the encrypted documents to the receiver. The encrypted version of the current BH-document is depicted in Figure 6.9:

<employees>
  <employee rank = 1 vstart = "01-06-2012" vend = "now"
tstart = "01-06-2012" tend = "UC">
    <EncryptedData
      Type = "http://www.w3.org/2001/04/xmlenc#Content"
      xmlns = "http://www.w3.org/2001/04/xmlenc#">
      <EncryptionMethod
        Algorithm = "http://www.w3.org/2001/04/xmlenc#aes128-cbc"/>
    </EncryptedData>
  </employee>
</employees>

Figure 6.9 (Continued)
Finally, the receiver uses the timestamp and the rank attribute, to make queries by the XQuery language on the encrypted current BH-document and encrypted history BH-document, to retrieve the required parts and decrypt them, instead of decrypting the entire encrypted current XML document and entire encrypted history XML document as follows:
• Query 1: retrieve the current managers:

```xml
for $s in doc("ECEmployees.xml")/employees/employee [ @ rank = 1 and @ vend = "now"]
return <managers> {$s} </managers>.
```

The above query is executed on the encrypted current BH-document (ECEmployees.xml), and the result is the first encrypted element. Hence, the receiver can decrypt the result and get the required information.

• Query 2: retrieve the previous managers who resigned on 31-12-2012:

```xml
for $s in doc("HEmployees.xml")/employees/employee [ @ rank = 1 and @ vend = "31-12-2012"]
return <managers> {$s} </managers>.
```

This query is executed on the encrypted history BH-document (HEmployees.xml).

• Query 3: retrieve all the previous employees:

```xml
for $s in doc("HEmployees.xml")/employees/employee [ @ rank = 2]
return <employees> {$s} </employees>.
```

• Query 4: retrieve all the current employees:

```xml
for $s in doc("ECEmployees.xml")/employees/employee [ @ rank = 2 and @ vend = "now"]
return <employees> {$s} </employees>.
```

From all these queries, it can be observed that, the proposed algorithm provides facilities to send queries to the XML databases, based on a combination of valid time and transaction time for time-stamping, and hence, can maintain the history data. Moreover, it is capable for decrypting the required parts of the encrypted databases, which are necessary to provide the
answer to the queries. This partial decryption helps in reducing the decryption time, and avoids the decryption of other parts thus enhancing the security.

6.4 RESULTS AND DISCUSSION

To encrypt and decrypt the XML databases, netbeans IDE 7 and the Apache XML Security have been used to encrypt the XML documents, encrypt the key used to encrypt the XML documents, and decrypt the XML documents. The parsing of the XML documents has been done, by using the DOM parser. To make XQueries, the XML documents have been stored in the eXist database. It is an open source native XML database management system, built by the XML technology. It is based on the XML data model to store XML data, and it has a highly efficient index-based XQuery processing. The Java application using the Apache XML security, and the eXist database run on the Windows 7 platform. The CPU is 2.20 GHz Pentium (R) Dual Core and 3GB RAM. The four XQueries mentioned in section 6.3 have been executed on the encrypted employees' XML documents stored in the eXist database, and a comparison between the times required for the partial and full decryption of the encrypted XML documents has been made. The results are depicted in Figure 6.10:

![Comparison between the Partial and Full Decryption of the Encrypted Employees' XML Documents](image-url)
In Figure 6.10, the time consumption is reduced, when the proposed algorithm is applied to the encrypted XML databases, to decrypt them partially.

Moreover, the proposed algorithm is applied on three XML Databases of Information System and Technology Department of Anna University:

1) IST Research Scholars (RSs).
2) IST UnderGraduates (UGs).
3) IST Employees (Emps).

For each encrypted XML database, after the execution of a set of XQueries on it, the percentage of the partial decrypted results of each XQuery is computed in relation to the full decryption. Hence, the average of the set of XQueries for each encrypted XML database is computed.

For the IST Research Scholars' (RSs) XML database, after applying the proposed algorithm, the current BH-document (CRSs.xml) will be as depicted in Figure 6.11:

```xml
<RSs>
  <RS Rollno="1116719192" vstart="01-01-2010" vend="now"
     tstart="01-01-2010" tend="UC">
    <Name>Mohamed</Name>
    <Area>XML security</Area>
    <Guid>A. Kannan</Guid>
    <Phone>3321</Phone>
    <Salary>9000</Salary>
  </RS>
  ...
</RSs>
```

**Figure 6.11 Proposed Current IST RSs' XML Database**
The following XQueries are executed on the encrypted temporal IST RSs' XML database (current and history BH-documents), and then the results are decrypted. Moreover, the same XQueries are executed on the encrypted conventional IST RSs' XML database after the decryption of the full document:

• Query 1: retrieve the RSs who joined on 01-01-2011:
  
  for $s in doc("ECRSs.xml")/RSs/RS [@vstart = "01-01-2011 "]
  return <RSs> {s} </RSs>.

• Query 2: retrieve the RS with roll no = "1116719192 ":
  
  for $s in doc("ECRSs.xml")/RSs/RS [@Rollno = "1116719192 "]
  return <RS> {s} </RS>.

• Query 3: retrieve the RSs who finished their Research:
  
  for $s in doc("EHRs.xml")/RSs/RS [@vend != "now "]
  return <RSs> {s} </RSs>.

• Query 4: retrieve the RSs who joined between 01-01-2009 and 01-01-2010, both days inclusive:
  
  for $s in doc("ECRSs.xml")/RSs/RS [@vstart <= "01-01-2010 "]
  and @vstart >= "01-01-2009 "]
  return <RSs> {s} </RSs>.

Figure 6.12 shows the difference between the partial decryption of the encrypted temporal IST RSs' XML database, and the full decryption of the encrypted conventional IST RSs' XML database, in the total time (compile time + execution time + decryption time) required to run the XQueries and get the results:
By applying the proposed algorithm, the running of the set of XQueries saves approximately 81% of the data from the decryption.

For the IST UnderGraduates' (UGs) XML database, after applying the proposed algorithm, the current BH-document (CUGs.xml) will be as depicted in Figure 6.13:

```xml
<UGs>
  <UG Rollno="1115619188" vstart="01-01-2010" vend="now" tstart="01-01-2010" tend="UC">
    <Name> Abdou </Name>
    <Semester> IV </Semester>
    <Courses>
      <Course> XML </Course>
      <Course> DB </Course>
      <Course> DM </Course>
      <Course> DS </Course>
      <Course> Cryptography </Course>
      <Course> N S </Course>
    </Courses>
    <GPA> 3.5 </GPA>
  </UG>
  ...
</UGs>
```

Figure 6.13 Proposed Current IST UGs' XML Database
The following XQueries are executed on the encrypted temporal IST UGs' XML database (current and history BH-documents) and on the encrypted conventional IST UGs' XML database, as done on IST RSs' XML database:

- **Query 1:** retrieve the UGs who joined on 01-01-2011:
  
  for $s$ in doc("ECUGs.xml")/UGs/UG [@vstart = "01-01-2011 "]
  return <UGs> {$s} </UGs>.

- **Query 2:** retrieve the UG with roll no= "1116519188 ":
  
  for $s$ in doc("ECUGs.xml")/UGs/UG[@Rollno = "1116519188 "]
  return <UG> {$s} </UG>.

- **Query 3:** retrieve the UGs who finished before 01-01-2010, including 01-01-2010:
  
  for $s$ in doc("EHUGs.xml")/UGs/UG[@vend <= "01-01-2010 "]
  return <UGs> {$s} </UGs>.

- **Query 4:** retrieve the UGs who joined between 01-01-2009 and 01-01-2010, both days inclusive:
  
  for $s$ in doc("ECUGs.xml")/UGs/UG [@vstart <= "01-01-2010" and @vstart >= "01-01-2009 "]
  return <UGs> {$s} </UGs>.

Figure 6.14 shows the difference between the partial decryption of the encrypted temporal IST UGs' XML database, and the full decryption of the encrypted conventional IST UGs' XML database, in the total time (compile time + execution time + decryption time) required to run the XQueries and get the results:
Figure 6.14  Comparison between the Partial and Full Decryption of the Encrypted IST UGs' XML Databases

By applying the proposed algorithm, the running of the set of XQueries saves approximately 73% of the data from the decryption.

For the IST Employees' (Emps) XML database, after applying the proposed algorithm, the current BH-document (CEmps.xml) will be as depicted in Figure 6.15:

```
<Emps>
  <Emp  ID = "11167" vstart = "01-01-2010" vend = "now"
       tstart = "01-01-2010" tend = "UC">
    <Name> Ali </Name>
    <Des> Director </Des>
    <Phone> 2341 </Phone>
  </Emp>
  ...
</Emps>
```

Figure 6.15  Proposed Current IST Emps' XML Database
The same set of XQueries, as done on RSs' and UGs' XML databases, are executed for employees on the encrypted temporal IST Emps' XML database and on the encrypted conventional IST Emps' XML database. Figure 6.16 shows the difference between the partial decryption of the encrypted temporal IST Emps' XML database, and the full decryption of the encrypted conventional IST Emps' XML database, in the total time (compile time + execution time + decryption time) required to run the XQueries and get the results:

Figure 6.16 Comparison between the Partial and Full Decryption of the Encrypted IST Emps' XML Databases

By applying the proposed algorithm, the running of the set of XQueries saves approximately 52% of the data from the decryption. Therefore, the proposed algorithm saves approximately 69% of the data from the decryption for all the three encrypted temporal XML databases. Figure 6.17 shows a comparison of the percentage of the decrypted data between the partial and full decryption of the three encrypted XML databases:
6.5 SUMMARY

In this chapter, a new time-stamp based algorithm for decrypting the encrypted XML documents partially is proposed. This proposed algorithm utilizes a new temporal model for XML representation, based on a combination of valid time and transaction time, through the tuple time-stamping. In addition to the time-stamp, one attribute of the key components of the encrypted elements is selected to distinguish among them. The time-stamp and the selected attribute guide the receiver to decrypt the required parts of the encrypted XML documents, instead of decrypting all the parts of them.

The new temporal model is proposed not only for the effective representation of the temporal XML, but also for providing facilities for encryption and decryption; i.e., the receiver uses the time-stamp and the selected attribute to retrieve the parts that must be decrypted, to contribute to the query results on the encrypted XML documents. Moreover, this model uses the valid time and transaction time for time-stamping, and hence, can maintain the history data efficiently. This model can be used to develop secure temporal applications.
CHAPTER 7

CONCLUSION AND FUTURE WORK

7.1 CONCLUSIONS

In this research work, new algorithms for mapping the XACML policies and rules into relational rules, and storing them in the form of rules in temporal relations, to ease the access control to the XML documents, are proposed.

Moreover, a new time-stamp based algorithm for decrypting the encrypted XML documents partially is proposed. This proposed algorithm utilizes a new temporal model for XML representation, based on a combination of valid time and transaction time, through the tuple time-stamping, and one selected attribute of the key components of the encrypted elements to distinguish among them.

7.1.1 Conclusions on the Mapping Algorithm

The new algorithms proposed in this work, for mapping the XACML policies and rules into relational rules help to represent the rules in relational format. Moreover, the rules designed for the temporal relations are useful to control the access to the XML documents effectively. These proposed algorithms help the users by relieving them from the complexity of the XACML structure, and also allow them to efficiently control the access to the XML documents, leading to reduction in their time and effort. From the experimental results obtained from this work, it is clearly proved that, the
proposed algorithms map the XACML policies and rules effectively. The major advantages of this proposed work are:

1. It makes the learning and understanding of the XACML policies and rules easier, and hence, it saves the users' time and effort.

2. It secures the access to the XML documents, which are stored in either native or relational databases, applying firing rules pertaining to XACML policies.

3. It applies the constraints of rules, and obligations and provides the response to an access request effectively.

### 7.1.2 Conclusions on the Decryption Algorithm

The new time-stamp based algorithm proposed in this work, for decrypting the encrypted XML documents partially guides the receiver to decrypt the required parts of the encrypted XML documents, instead of decrypting all the parts of them. This proposed algorithm utilizes a new temporal model for XML representation, based on a combination of valid time and transaction time as the tuple time-stamping. Moreover, one attribute of the encrypted elements is selected as one of the key components to distinguish among them. The proposed algorithm not only reduces the decryption time, but also protects the security of the data.

The new temporal model is proposed not only for the effective representation of the temporal XML documents, but also for providing facilities for encryption and decryption; i.e., the receiver uses the time-stamp and the selected attribute to retrieve the parts that must be decrypted, to contribute to the query results on the encrypted XML documents. Moreover, this model uses the valid time and transaction time for time-stamping, and
hence, can maintain the history data efficiently. This model can be used to develop secure temporal applications.

7.2 FUTURE WORK

Many future enhancements to this work are possible. Among them, proposal of techniques for Detecting XACML policies and conflicts with respect to a combination of a target, a subject, a resource, an action, and other environment attributes, is an important suggestion for future work. In addition, mobile agents can be deployed to improve the access control by monitoring the applications or policies.
REFERENCES


169


LIST OF PUBLICATIONS

International Journals


International Conferences


8. **Abd El-Aziz, AA & Kannan, A** 2013, ‘JSON Signature’, Proceedings of the Third International Conference on Computational Intelligence Technology-CIIT 2013, Mumbai, India, pp. 1-6. The proceeding will be forwarded to be indexed in Elsevier.