UNIVERSITY OF CALIFORNIA, IRVINE

Querying Encrypted XML Documents

THESIS

submitted in partial satisfaction of the requirements for the degree of

MASTER OF SCIENCE

in Electrical and Computer Engineering

by

Ravi Chandra Jammalamadaka

Thesis Committee:
Professor Stephen Jenks, Chair
Professor Sharad Mehrotra,
Professor Philip Sheu

2004
The thesis of Ravi Chandra Jammalamadaka is approved:

Committee Chair

University of California, Irvine
2004
Dedication

To my family
# TABLE OF CONTENTS

LIST OF FIGURES vi

LIST OF TABLES viii

ACKNOWLEDGMENTS ix

ABSTRACT OF THE THESIS x

## 1 Introduction

## 2 Background

- **2.1 XML data model** ................................................................. 4
- **2.2 Cryptography** ................................................................. 5
  - **2.2.1 Basic Cryptography model** ........................................ 6
- **2.3 XML Encryption** ............................................................... 7
  - **2.3.1 Design Philosophy** .................................................. 8
  - **2.3.2 Element-wise encryption** ........................................ 8
- **2.4 Background on Hacigumus Scheme** .................................... 9

## 3 Encryption primitives for XML documents 13

- **3.1 Encrypt Value Primitive $E_V$** ........................................ 14
3.2 Encrypt Tag Primitive $E_T$ ........................................... 14
3.3 Encrypt structure primitive $E_S$ ........................................... 15
3.4 Conflicting primitives ............................................................ 18

4 Encrypted XML storage model .................................................. 20

5 Ancillary Information ................................................................. 24
  5.1 Subtree without multivalued operators .................................... 24
  5.2 Subtree with multivalued operators ....................................... 26
  5.3 Content of the node Ancillary(parentid) .................................. 28

6 Security evaluation ................................................................. 30

7 Query Translation ................................................................. 36
  7.1 Mapping the tree structure of the pattern tree ......................... 38
  7.2 Mapping predicates .......................................................... 41

8 Experiments ............................................................................ 43

9 Related Work ........................................................................... 49

10 Conclusions ............................................................................ 52
LIST OF FIGURES

2.1 Example XML Schema and an XML document confirming to it .......................... 5
2.2 Basic Cryptography Model ............................................................................. 7
2.3 XML document encryption ............................................................................. 9
2.4 Partitioning of the attribute Age domain ..................................................... 11

3.1 Effect of the primitives on the original document ........................................ 16
3.2 Conflicting primitives ..................................................................................... 18

4.1 Schema mapping .............................................................................................. 21
4.2 $etsub^m$ .......................................................................................................... 22
4.3 Effect of the $E_S$ primitive ............................................................................ 23

5.1 $etsub^e$ ........................................................................................................... 27
5.2 Construction of the partitioning information .................................................. 28

6.1 Single dimensional partitioning ...................................................................... 32
6.2 Multidimensional partitioning ........................................................................ 33
6.3 Employee Schema ........................................................................................... 33

7.1 Example of a Pattern ....................................................................................... 37
7.2 Query Translation ............................................................................................ 37
7.3 Resolved pattern .................................................. 38
7.4 Effect of the $E_S$ primitive on the pattern ......................... 40
7.5 Effect of the $E_V$ primitive on the pattern ......................... 40

8.1 Schema of the XML documents used in the experiments ............ 44
8.2 Server side performance for $Q_2$ .................................. 47
8.3 Server side performance for $Q_1$ .................................. 48
LIST OF TABLES

2.1 Employee Relation ........................................ 10
2.2 Encrypted Employee Relation ............................. 11
5.1 Partitioning the $subtree(\text{Author})$ domain ........ 26
8.1 Query set used in the experiments ....................... 44
8.2 Client side performance of the estubs for Query $Q_1$ 44
Acknowledgements

I am grateful to Prof. Stephen Jenks for his time and patience to ensure the completion of this thesis.

I thank my advisor Prof. Sharad Mehrotra for this constant support and motivation during the course of this research.

I would like to thank Prof Philip Sheu for agreeing to be in my thesis committee and providing me helpful comments and suggestions.

I am thankful to all my group members, particularly Dawit Yimam Seid, Iosif Lozaridis, for their helpful discussions and constant encouragement.

I want to express my deep gratitude to my family, for their love and sacrifice, which helped me cross many hurdles and reach this far.
ABSTRACT OF THE THESIS

Querying Encrypted XML Documents

By

Ravi Chandra Jammalamadaka

Master of Science in Electrical and Computer Engineering

University of California, Irvine, 2004

Professor Stephen Jenks, Chair

This thesis proposes techniques to query encrypted XML documents. Such a problem predominantly occurs in “Database as a Service” (DAS) architectures, where a client may outsource data to a service provider that provides data management services. Security is of paramount concern, as the service provider itself is untrusted. Encryption offers a natural solution to preserve the privacy of the data owner’s data. The challenge now is to execute queries over the encrypted data, without decrypting them at the server side. The first contribution is the development of the security mechanisms on the XML documents, which help the client to encrypt portions or totality of the XML documents. Techniques to run SPJ (Selection-projection-join) over encrypted XML documents are analyzed. A strategy, where indices/ancillary information is maintained along with the encrypted XML documents is exploited, which helps in pruning the search space during query processing.
Chapter 1

Introduction

Over the past decade, with the explosive growth of the internet, networking, and computing technologies, the software industry has witnessed the emergence of the software as a service paradigm whereby clients, instead of installing, maintaining and running software on their computer systems, can purchase the software usage on a “rent an application” basis from software service providers. Motivated by the software as a service, recent database research has explored the viability of the “database as a service” (DAS) paradigm [1, 2, 16, 24] in which clients store their databases on a remote service provider that offers full-blown data management services (storage, query, administration, backups, recovery, etc.). Many research and technological challenges in supporting DAS have been identified the primary of which is the issue of data privacy and security. Since in the DAS model, the client’s data, some of which may be sensitive, resides at the service provider outside the security perimeter of the client, mechanisms to prevent data misuse have to be developed. Encryption is the natural answer, but that raises a fundamental (and difficult) challenge of how to execute queries over encrypted data. Many innovative proposals addressing this
challenge have recently emerged [2, 16] in which the client and server collaboratively process queries; as much of the query as is possible to execute without decrypting is executed in the untrusted domain, and when further processing is not possible without decryption, the data is brought to the trusted side and converted to plaintext. Techniques to handle many different classes of queries (selection, joins, aggregation) have been previously studied. Much of this has considered outsourcing in the context of the relational data model.

This thesis focuses on outsourcing data in the context of the XML databases. XML data, beside content, also consists of a rich internal structure. A client, besides (or instead of) hiding the attributes and elements of the XML document may also desire to hide relationships among nodes. We develop simple, yet powerful, security primitives using which clients (data owners) can specify a rich class of privacy policies for XML data. The security policies of the client are cryptographically enforced and the encrypted documents are hosted at the server. To facilitate query processing, motivated by the strategy in [2], ancillary information along with the encrypted XML documents is stored. Instead of exactly utilizing the approach in [2], we develop a novel strategy of multi dimensional partitioning which overcomes some of the privacy limitations of the approach in [2]. Translation of an original XML query Q into two queries, one query to be executed at the server and one query to be executed at the client is then shown.

The remainder of the thesis is organized as follows: In chapter 2 an overview of relevant background material is presented. In chapter 3 we introduce the privacy
primitives using which flexible privacy policies can be specified. In chapter 4 and 5 we describe the encrypted XML storage model for server. In chapter 6, we show how our techniques withstand against the popular attack models in the security literature. In chapter 7 we deal with client side query translation to an equivalent server side query. In chapter 8 we study with the performance analysis of our proposed techniques.
Chapter 2

Background

2.1 XML data model

XML markup language is a subset of the Standard Generalized Markup Language (SGML). XML documents typically follow a tree-like structure, containing nested elements delimited by start and end tags. An XML document is well-formed if every element has a start and end tag inside the element, which contains the start tag. DTDs (Document Type Definition) describe the format of the XML document confirming to them. They can be included with the XML documents or externally referenced. DTDs are primarily composed of elements, attribute and entity declarations. Element declaration states the name of the element and its allowed content. The allowed content could contain subelements and their cardinality. Cardinality is usually described using the operators "*" (zero or more occurrences), " +" (one or more occurrences), " ?" (optional) and " | " (or). Attribute declarations specify for each element the list of attribute names, which describe the properties of the element. XML schema language address the limitations of DTDs. Most notable extensions are
stronger data typing, and size restrictions on data values. In this paper we will assume non-recursive XML schemas or DTD’s and represent them as labeled trees by creating a node for every element and attribute. Fig 2.1 gives an example of an XML schema and an XML file corresponding to it. The schema describes details of papers that were submitted to a conference.

2.2 Cryptography

Security of computer systems deals with prevention of unauthorized access to hardware and software resources. Maintaining data confidentiality is a paramount concern
for business organizations, because lack of it could lead to major financial losses. However an authorized user can still tap into physical device (ex. a wire) and gain control of unauthorized information. When information is disseminated in unsecure domains, the science of cryptographic techniques is used to protect confidential information and authenticate users to other users. Cryptography works on the basic principle of transforming clear intelligent text (henceforth, referred to as plaintext) to an unintelligent cryptic form (henceforth, called the ciphertext). This process is called encryption. If an unauthorized user gains access to ciphertext, information about the plaintext is not leaked. An authorized user can decipher the ciphertext back to plaintext with the knowledge of secret information such as a key. This process is called decryption.

### 2.2.1 Basic Cryptography model

Fig 2.2 shows the basic cryptographic model, which has been taken from [27]. Block E denotes the encryption function \( E_{K_e} : P \rightarrow C \), where \( P \) is the plaintext message space and \( C \) is the ciphertext message space. A message \( m \in P \) is encrypted to ciphertext \( c \in C \), using the encryption key \( K_e \). This ciphertext is transmitted over an unsecure channel to a destination where it is deciphered. The Block D denotes the decryption function \( D_{K_d} : C \rightarrow P \), which transforms cipher text \( c' \in C \) to a plaintext \( m' \in P \), using the decryption key \( K_d \). The property \( D_{K_d}(E_{K_e}(m)) = m \) should hold for all \( m \in P \).

Block CA denotes the cryptanalyst (i.e. unauthorized user) who with the help of some information (SI) and access to cipher text \( c \) tries to infer information about
plaintext $m$. The Side information (SI) could include language statistics, context for ongoing communication and previous ciphertexts or portion of plaintext. Since the cryptanalyst does not have knowledge of the decryption key $K_d$, the security of this system is measured by the difficulty in breaking the key $K_d$.

### 2.3 XML Encryption

XML has become the de facto standard for exchanging data for web applications. Large XML repositories are becoming a common place in the corporate world. Encrypting XML is a method widely used for securing XML documents. XML encryption has been studied in the context of access control and for secure transmission of XML documents through an unsecure channel. This section describes the W3c consortium recommended standard for XML encryption [26]. The encrypted data is represented back as XML data. The first part of the section primary design require-
ments on which the XML standard is based upon and in the latter part of the section describes the techniques to implement this philosophy.

2.3.1 Design Philosophy

1. The granularity of encryption is an element.: XML document has a nested element structure. To provide partial encryption of the XML document, individual elements (both the tag and their contents) can be encrypted, which is the granularity that XML provides. For encrypting the total XML document, the root element needs to be encrypted.

2. Encrypted information must be separated from the unencrypted information. This is necessary to ensure that encrypted information can be filtered out of the XML document for decryption to take place.

3. Encrypting elements that have already been encrypted, should be allowed: This process is termed super-encryption. This is typically performed for generating XML signatures and enforcing access control on XML documents

4. Encrypted Key information should be conveyed to the recipient: The need for this is obvious as without appropriate information, the recipient may not be able to decrypt the information.

2.3.2 Element-wise encryption

This section describes the element wise encryption technique recommended by the W3C consortium, which adheres to design philosophy for XML encryption described above. When an element is encrypted, its tag value, together with its content is
streamlined into a text and this text is encrypted. The encrypted representation of the
element is placed as a content of an Encrypted node. Consider the XML documents in
Fig 2.3a, which describes the details about a sale of a book. One security requirement
could be encryption of the element <Credit_Card_Information>. Fig 2.3b shows
the final encrypted XML document. Node cipherdata now contains the encrypted
representation of the node <Credit_Card_Information>, which is a child of the node
EncryptedData. The Attributes of the node encrypted data, store information about
the encryption algorithm used and keysize.

2.4 Background on Hacigumus Scheme

This section summarizes the techniques proposed by Hacigumus et al [2] for executing
SQL queries over encrypted relational databases. For each client side relation R( A₁,
A₂, ... Aₙ) an encrypted relation is stored at the server side, with the following
schema:

\[
R^S(\text{tuple}, A_1^S, A_2^S, \ldots A_n^S)
\]
<table>
<thead>
<tr>
<th>Name</th>
<th>Age</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>30</td>
<td>60K</td>
</tr>
<tr>
<td>Jim</td>
<td>56</td>
<td>100K</td>
</tr>
<tr>
<td>Mary</td>
<td>59</td>
<td>30K</td>
</tr>
</tbody>
</table>

Table 2.1: Employee Relation

Attribute *etuple* stores the encrypted string of the tuple that belongs to relation R. For every attribute $A_k$ in R, an index $A_k^*$ is maintained which is used during query processing. The values stored in these indices will be explained shortly.

**Partition function:** Partition function maps the domain of a attribute $R.A_k$ to a set of non-overlapping partitions or buckets $\{p_1, p_2 \ldots p_t\}$. $P_j$.high and $P_j$.low represents the highest and the lowest value contained in the partition. Formally, $\text{partition}(R.A_k) = \{p_1, p_2 \ldots p_t\}$. Consider the employee relation illustrated in table 2.1, which stores records of employees working for a certain company. For instance, the domain of attribute age in the employee relation can partitioned as $[0,25],[25, 50]$, $(50, 75]$, $(75, 100]$. Any popular histogram technique such as equi-width, equi-depth, ... etc could be used for partitioning the attribute domain space. Care should taken so that one single value in the domain is mapped to only one partition, which may not be the case in equi-depth.

**Identification function:** The identification function $\text{ident}_{R.A_i} : P_j \rightarrow n$, where $n \in \mathbb{Z}^+$ assigns an unique identifier for every partition $P_j$ of an attribute $A_i$. Figure 2.4 illustrates the identifiers assigned to the partitions of age attribute.
Mapping function The Mapping function $Map_{R,A_k} : v \rightarrow ident_{R_{A_k}} (P_j)$, where $v \in P_j$ maps a value $v$ of an attribute $A_k$ to the identifier of the partition to which it belongs.

$A_k$ stores the values of $Map_{R_{A_k}} (v)$ for all the values $v \in A_k$. For instance, John is a 30 year old and his age maps to partition $(25,50]$ which is identified by 12, therefore 12 is stored in $Age^*$ for John’s tuple. The server side representation of the employee relation in table 2.1 is shown in table 2.2. Not all attributes are needed to have partition indices stored at the server side, only attributes which likely to be used for selections and joins by queries are created an index.

The query translation procedure will now be briefly explained. The client utilizes the partitioning strategy to map the query conditions to their equivalent server side conditions. The server executes this less precise query and sends back the results back to the client, which are later decrypted and the false positives filtered out. For instance, a client side query
SELECT Age FROM R WHERE Age BETWEEN 20 and 45

is transformed to:

SELECT Age^e FROM R^s WHERE Age^s = 6 or Age^s = 12

This scheme dealt only with SPJ (selection-projection-join) queries. In [21] the authors show how LIKE queries can be supported in the DAS model. For more information regarding the Hacigumus scheme, the reader is referred to [2].
Chapter 3

Encryption primitives for XML documents

This section describes primitives using which a client can specify security requirements over both the structure as well as the content of XML documents. The client may not wish to protect the XML data in its entirety and may choose instead to protect only the sensitive information. For instance, in an XML document containing the name, address and credit card information of customers, the client may wish to protect the credit card information, but leave the address information unencrypted. There are performance benefits of partial encryption due to reduction in amount of decryption at the client side. It will be shown later that partial encryption has benefits for query processing.

Specifically three primitives are explored, using which the client can specify fairly complex security policies over the structure, content and the metadata of the XML documents. These primitives could be applied at both the individual documents or at
the *schema level*, in which case they are applicable to all the XML documents which conform to the XML schema. This thesis considers only the privacy specifications at the *schema level*.

### 3.1 Encrypt Value Primitive $E_V$

**Syntax:** $E_V(n)$, where $n \in E$

Let $E$ be the set of nodes of an XML document $D$. $\text{subtree}(n)$ denotes the tree structure induced by the descendents of the node $n$. $\text{subtree}(n)$ represents both the tag and the content of node $n$. When $E_V(n)$ primitive is specified, both the content and the tag information of the node $n$ is encrypted. The $\text{subtree}(n)$ is replaced by an encrypted node. This is similar to the approach followed in [3,4] for securing XML documents. Fig 3.1b shows an example of the changes due to specifying $E_V(\text{author})$ on the XML document in Fig 3.1a. $\text{subtree}(n)$ is serialized as text and encrypted, this encrypted string is stored as the value of the encrypted node. The tag value of the encrypted node can be arbitrarily assigned as long it is stored as metadata at the client. Information regarding the descendents of the node $n$ is encrypted. Fortunately, since XML documents are accompanied with an DTD/XML schema, the documents can be reconstructed at the client.

### 3.2 Encrypt Tag Primitive $E_T$

**Syntax:** $E_T(n)$, where $n \in E$
This primitive encrypts the content of the tags of the XML documents. When specified on nodes which are not the leaf nodes, the primitive cascades down all the way down to the leaf level and encrypts the tags which belong to \( \text{subtree}(n) \). There could be scenarios where protecting the metadata (i.e. the start and the end tags) achieves the desired privacy and the client does not need to encrypt the content. Clearly, then all the queries can be executed at the server side, since the data is unencrypted. For instance, consider the element \(<\text{Number of years spent in jail}>\), the value of this element is not important, but the tag of this element gives away potentially controversial or damaging information. \( E_T \) primitive is also likely to be used in cases where the client wants to protect the internal structure of \( \text{subtree}(n) \), but hide both tag value and the content of all the nodes that belong to the \( \text{subtree}(n) \). This will be motivated by an example. The client specification \( E_V(\text{Author}) \) destroys the structure of XML document as the \( \text{subtree}(\text{Author}) \) was replaced with an encrypted node. To preserve the structure of the \( \text{subtree}(\text{author}) \), another option to secure the \( \text{subtree} \) is to encrypt all the tags and the leaf node content belonging to it. This is achieved with \( E_T(\text{Author}) \) and the \( E_V \) primitive imposed on all the leaf nodes belonging to \( \text{subtree}(\text{Author}) \). Such a specification could have better performance but does not offer the same security as encrypting the whole subtree. Fig 3.1c illustrates the effect of the \( E_T(\text{Author}) \) primitive on the XML document in Fig 3.1a.

### 3.3 Encrypt structure primitive \( E_S \)

**Syntax:** \( E_S(\text{parentNode}, \text{childNode}) \) where \( \text{parentNode} \) and \( \text{childNode} \in \text{E} \) and node \( \text{parentNode} \) is parent of node \( \text{childNode} \)
Figure 3.1: Effect of the primitives on the original document
In some situations, it becomes more useful in hiding the relationship shared between nodes than explicitly encrypting either the tags or content. The client could prefer to publish the information publicly, as long as the relationship between some nodes is hidden. Consider the XML schema introduced in Fig 1. For blind reviewing\textsuperscript{1} to be successful, only the relationship between the author and the paper nodes needs to be hidden. The community for a conference is well known in most cases. The client will agree to publish the author name and email information as long as the relationship between the paper node and the author node is hidden. $E_S(parent\ Node,\ child\ Node)$ primitive achieves the above objective by fragmenting the original XML document. Fig 3.1d shows the effect of $E_S(paper,\ author)$ primitive on a XML document confirming to the XML schema introduced in fig 3.1a. If fragmentation was not followed, the other option would have been either to encrypt the $subtree(Author)$ or encrypt $subtree(Paper)- subtree(Author)$. By fragmenting the document, queries such as “find the email of Jeff Ullman” can be run on the documents stored at the server. Any query that can be answered by $subtree(Author)$ can be executed at the server. Previous XML access control models have treated only elements or attributes as objects. In this thesis, both the edges and the nodes (elements and attributes) are treated as objects. Hiding only the relationship but not the content is not to best of our knowledge explored before. The flexible security model developed in this paper could also be used in other contexts where XML access control is necessary.

\textsuperscript{1}A process used by some conferences to keep the name of the author secret from the reviewer and vice versa
### 3.4 Conflicting primitives

We will define primitives that cannot co-exist or privacy specifications from these primitives cannot be satisfied at the same time.

**Definition 6.1:** Two primitives $E_S(\text{parentNode}, \text{childNode})$ and $E_V(n)$ are conflicting if either of $\text{parentNode}$ or $\text{childNode} \in \text{subtree}(n)$.

This situation is illustrated in Fig 3.2. If $E_V(n)$ is imposed first, it will replace the $\text{subtree}(n)$ with an encrypted node (as described in Section 3.1) and nodes $\text{parentNode}$ and $\text{childNode}$ will cease to exist. Therefore, $E_S(\text{parentNode}, \text{childNode})$ primitive cannot be satisfied. If $E_S(\text{parentNode}, \text{childNode})$ is imposed first, then $E_V(n)$ cannot be fully satisfied as $\text{subtree}(\text{childNode})$ is detached from the original schema. We will resolve this situation by giving $E_V(n)$ the precedence and ignoring the $E_S$ primitive.
**Definition 6.2:** Two primitives $E_V(n)$ and $E_T(n_1)$ are conflicting if node $n_1$ is the ancestor of node $n$ or vice versa.

If $n$ is the ancestor to $n_1$, then the $E_T(n_1)$ primitive will be ignored as $n_1$ will cease to exist after the $E_V$ is imposed. If $n_1$ is the ancestor to $n$, then we will impose the $E_V(n)$ primitive, by replacing $subtree(n)$ with an encrypting node and we will also encrypt all the tags from $n_1$ to the encrypting node. Intuitively, the $E_T$ primitive is satisfied as much as possible after $E_V$ primitive is specified.
Chapter 4

Encrypted XML storage model

The previous section developed primitives on the XML schema using which the client can specify the security policy on the XML schema. Encryption strategies to realize the client’s security policy were also specified. Our objective, however, is to develop an encrypted XML representation that allows query processing. This section develops a mapping that takes as input, the XML schema as well as the security policy specified using the primitives developed in chapter 3 and outputs a server side XML representation \(^1\).

In our approach similar to [2], the client stores ancillary information at the server to facilitate query processing. However, the specific ancillary information stored differs from [2] and this will become clear in the following sections.

\(^1\)The server can store the encrypted XML documents using a native XML database or an RDBMS using the translation techniques in [11][12]. Our techniques are independent of the type of the database employed at the server.
Input:
XML schema $S$ of the unencrypted XML documents
$E_{prim} = \{ E_1, E_2, \ldots, E_p \}$ where $E_i \in \{ E_S, E_V, E_T \}$

Mapping:
1. For every $E_V(n)$ primitive $\in E_{prim}$
   
   - Replace $\text{subtree}(n)$ with $\text{estub}^m$

2. For every $E_S(\text{parentNode}, \text{childNode})$ primitive $\in E_{prim}$
   that does not conflict with any $E_V \in E_{prim}$
   
   - Fragment the schema into two different trees and create
     node id as the child of $\text{parentNode}$ and nodes $\text{parentid}$
     and pid as the children of $\text{childNode}$

3. For every $E_T(n_1)$ primitive
   
   - Encrypt the tag values of all the nodes $\in \text{subtree}(n_1)$ unless
     the node is an encrypted node obtained from step 1.

Figure 4.1: Schema mapping

Primitives are handled first. When the client specifies $E_V(n)$ primitive ideally he
desires to replace $\text{subtree}(n)$ with an encrypted node. In the approach proposed in
this thesis the $\text{subtree}(n)$ is replaced by $\text{estub}^m$, whose structure is shown in Fig 4.2.
Let $L = \{ L_1, L_2, \ldots, L_N \}$ be the set of leaf nodes that are affected by the specification
of $E_V(n)$ primitive, i.e all the leaf nodes that belong to $\text{subtree}(n)$. Node $E(L_1 \ldots L_N)$
stores the the encrypted string of the concatenation of all the leaf node values.
Nodes $\text{Ancillary}^1(L_V)$ and $\text{Ancillary}^2(L_{nv})$ store the ancillary information required
for querying on the leaf nodes in $L$. Let $L_V$ be the set of leafNodes which do not
have a "*" or "+" from the path from the node where the $E_V$ primitive is imposed,
to themselves. Let $L_{nv}$ be the other leaf nodes which have a "*" or "+" in the path
from the node where the $E_V$ primitive is imposed to themselves. For the nodes in $L_{nv}$
their cardinality is not known. The content of this ancillary information is explained
in the next section.

$E_S$ primitives that do not conflict with the $E_V$ primitives are handled next. $E_S(\text{parentNode}, \text{childNode})$
fragments the original document tree structure into two different trees, one that ends at \textit{parentNode} and another that is rooted at \textit{childNodes}. This implies isolation of subtree rooted at \textit{childNodes} from the original document. Two nodes \textit{id} and \textit{parentid} are created, children of \textit{parentNode} and \textit{childNodes} respectively. When \textit{id} = \textit{parentid}, it implies that both these nodes belong to the same document. Both the \textit{id} and \textit{parentid} nodes are self generating attributes and one of them needs to be encrypted to enforce the \textit{Es} primitive and hide the relationship between \textit{parentNode} and \textit{childNodes}. The content of \textit{parentid} is encrypted as the node \textit{parentNode} could potentially have more than one child. Another node \textit{Ancillary(parentid)} is created, which stores ancillary information required to process a join operation with between the nodes \textit{parentNode} and the \textit{childNodes} at the server. The content of \textit{Ancillary(parentid)} and its use is explained in the next section. Fig 4.3 shows the effect of the \textit{Es} primitive.
Figure 4.3: Effect of the $E_S$ primitive

We have previously shown how $E_T(n)$ primitive encrypts tag value and this primitive cascades down to the leaf level unless it encounters an already encrypted node. No other changes are required to this mapping to facilitate query processing.
Chapter 5

Ancillary Information

This chapter explains the content of ancillary information stored at the server to support query processing. The ancillary information stored when the $E_V$ primitive is imposed [i.e. the content of the nodes $Ancillary^1(L_V)$ and $Ancillary^2(L_{rev})$] is discussed first. This is discussed under two situations a) when the subtree($n$) affected by the $E_V$ primitive does not contain multivalued operators (i.e. “*” or “+” operator) and b) when the subtree($n$) affected by the $E_V$ primitive contains multivalued operators. The two cases are handled in Section 5.1 and Section 5.2 respectively. Section 5.3 discusses the ancillary information stored for the $E_S$ primitive.

5.1 Subtree without multivalued operators

In this situation ancillary information is only stored in the node $Ancillary^1(L_V)$. Let the set $L = \{ L_1 \ldots L_N \}$ be the set of leaf nodes belonging to subtree($n$). Since there are no multivalued operators in subtree($n$) set $L_v( all the leaf nodes which
do not have a '*' or '+' from the path from the node where the $E_V$ primitive is imposed, to themselves) is equal to $L$. Let $\text{dom}(L_i)$ represent the domain of the leaf node $L_i$. Let $\text{dom}(L) = \text{dom}(L_1) \times \text{dom}(L_2) \times \ldots \text{dom}(L_N)$ be the cartesian product of all the domains of leaf nodes elements in $L$. The domain of $L$ can be viewed as a $N$-dimensional space where each leaf node $L_i$ corresponds to a dimension. This $N$-dimensional space is partitioned into a set of partitions and associate a random identifier for each partition. The partitions should cover the whole domain and should not overlap. Any popular multi dimensional histogram\cite{22} based technique could be used to partition this multi-dimensional space as long as care is taken that a point is mapped only to one partition. The choice of the multi-dimensional partitioning strategy has implications to security. Section 6 will describe why equi-depth is the best policy. This partitioning information is stored at the client side and kept secret from the server.

Table 5.1 shows an example multi dimensional partitioning strategy. Note that string dimensions (FN, LN, Email) have to be first mapped to an integer domain using hashing techniques. The unencrypted leaf node values of $\text{subtree}(n)$ are points in the multi-dimensional space corresponding to $\text{dom}(L)$. The partition identifiers of these points are stored as the content of the node $\text{Ancillary}^1(L_V)$ along with the encrypted information i.e. $E(L_1 \ldots L_N)$.

The ancillary information maintained in this thesis is used for processing Selection-projection-join queries. A class of queries not handled here are the LIKE queries. In \cite{21} the authors propose an $n$-gram based approach to handle LIKE queries. A similar technique can be incorporated in this model, although it is not dealt here.

Note that we could have used the strategy in \cite{2} where the domain of every leaf
node \( L_i \in subtree(n) \) is partitioned and partition identifier is stored as the ancillary information. If we were to follow this strategy then \( subtree(n) \) will be replaced by the \( estub^m \) which is shown in Fig 5.1. \( P(L_i) \) stores the partition identifier for leaf node \( L_i \). There are two reasons for adopting \( estub^m \) instead of \( estub^s \) a) Multi dimensional partitioning is more secure than the single dimensional partitioning. We will show why this is the case in chapter 6 and b) \( estub^s \) cannot handled the case when \( subtree(n) \) contains a multivalued operators (i.e. "*" or "+").

### 5.2 Subtree with multivalued operators

In this situation ancillary information on both the nodes \( Ancillary^1(L_v) \) and \( Ancillary^2(L_{nv}) \) is stored. The contents of set \( L_v \) and \( L_{nv} \) have been explained in Chapter 5. All the nodes in \( L_v \) can now be treated as a dimension and this space can be multi dimensionally partitioned using the strategy we discussed above. The partition identifier is stored as the content of \( Ancillary^1(L_v) \).

Now we will explain the content of the \( Ancillary^2(L_{nv}) \) node. For nodes in \( L_{nv} \) the
cardinality of the leaf nodes is not known. It is difficult to use the multi dimensional partitioning strategy for set based elements since the cardinality of the leaf nodes keeps changing. We could have mined the maximum cardinality of the leaf nodes and used it to partition the content space, but such an approach is not scalable. For all leaf nodes in $L_{nv}$ we will partition their multiple values single dimensionally and store the string concatenation of all the partitions as the content of the $Ancillary^2(L_{nv})$. Fig 5.2 illustrates the construction for the content of the nodes $Ancillary^1(L_V)$ and $Ancillary^2(L_{nv})$. Queries on the leaf nodes in $L_{nv}$ are translated into LIKE queries over the $Ancillary^2(L_{nv})$ node. This translation would be explained in more detail in Section 7.
5.3 Content of the node \textit{Ancillary}(\textit{parentid})

To preserve the privacy of the $E_5(\text{parentNode, childNode})$ primitive, a new node \textit{Ancillary}(\textit{parentid}) was introduced as the child of the node \textit{childNode}. This section will explain the content of the node \textit{Ancillary}(\textit{parentid}).

\textbf{Definition 9.1:} Given a \textit{parentNode} $P_1$ and any two \textit{childNodes} $C_1$ and $C_2$, let the function $\text{Prob}(n_1, n_2)$ give the probability that $n_1$ is the parent of $n_2$, then the privacy of $E_5(\text{parentNode, childNode})$ is only preserved if $\text{Prob}(P_1, C_1) = \text{Prob}(P_1, C_2)$.

The above definition explains the requirement for the preservation of the privacy of the $E_5$ primitive. Two groups of trees can be easily stored at the server. To execute a path query which traverses the edge between \textit{parentNode} and \textit{childNode}, a join needs to be performed between \textit{parentNode} and \textit{childNode}. The join clearly cannot take place at the server, because that would compromise the primitive. The only possible
way is to ship all the documents rooted at $childNode$ node to the client. This clearly
is infeasible solution for large databases. A similar problem was addressed by the
PIR( private information retrieval) community [13][14]. In PIR, the client wishes to
retrieve a record from a database which belongs to the server, without revealing any
information about record that has been retrieved. The solutions involved using non
colluding duplicate severs or secure coprocessors which scanned the whole database
to fetch the required data. Both these solutions are impractical in our setting.

Our technique is to ship only a subset of documents rooted at $childNode$. The client
can control the number of documents being shipped and this becomes a security
parameter. Recall the creation of the nodes $id$ and $parentid$. The content of the
node $id$ is kept unencrypted and the content of the node $parentid$ is encrypted. The
domain of the node $id$ is single dimensionally partitioned as explained before and
the partition identifier stored as the content of the node $Ancillary(parentid)$. The
metadata regarding the $id$ partitioning can be stored with the server. When the join
has to take place between the $parentNode$ and the $childNode$, the partition identifier $p$
where the content of the $id$ node maps to is found, and the XML documents rooted at
$childNode$ and having the content of the node $Ancillary(parentid)$ to be $p$ are shipped
to the client. The client can now decrypt all the $parentid$ nodes and execute the join
operation.
Chapter 6

Security evaluation

This section analyzes the relative security merits of multi dimensional partitioning scheme and the single dimensional partitioning scheme. It is assumed that encrypted information stored at the server is secure and the adversary only learns information from the ancillary information (partitions). For simplification the service provider is assumed to be the adversary, but in reality other agents could collude with the service provider. The reader should notice that in reality, the relative security offered by $estub^s$ and $estub^m$ which use the single dimensional and multi-dimensional partitioning respectively is being compared. Two security models semantic security and $K$-anonymity are used to evaluate both the partitioning strategies, but first the adversary prior knowledge of the encrypted database is described.

- Schema S of the unencrypted XML documents. This implies that the adversary has exact knowledge of all the leaf nodes that belong to an $estub^d$, where $d \in \{s, m\}$. With some of the nodes being kept in the plaintext, the adversary can guess an the approximate structure of the XML documents. To strengthen the
arguments the attacker is assumed to have the full knowledge of the schema.

- Additionally for the estub* case we will assume that that adversary also knows the corresponding partition index $P(L_i)$ for every node $L_i$.

- The probability density function $D(i)$ for every leaf node $L_i \in estub^d$, where $d \in \{s, m\}$. Even here the adversary could know the approximate domain distribution. But for simplicity and to strengthen our arguments the adversary is assumed to have the exact knowledge.

The semantic security model also called Indistinguishability first introduced in [29] is described now. The lack of semantic security in the single dimensional partitioning is described in [27][28].

**Semantic Security:** Given two plaintext messages $P_1$ and $P_2$ and one ciphertext $C = E_K(P_i)$ where $i \in \{1, 2\}$ and $E_K$ is the encryption function with key $k$, if the adversary can guess $i$ with probability $1/2 + \delta$, where $\delta > 0$, then the encryption function is not semantically secure.

The above definition is similar to the one defined in [18]. Intuitively, if the encryption scheme leaks any partial information, then such a encryption scheme is not partially secure. In [17] the authors model semantic security as an interactive game between the client and the server. We will now describe the game, making necessary modifications to suit our context for XML data.

- The server creates two unencrypted subtrees, $subtree(n_1)$ and $subtree(n_2)$ and sends them to the client.

- The client encrypts the two subtrees and sends the estub^d's, where $d \in \{s, m\}$ back to the server.
Figure 6.1: Single dimensional partitioning

- The client chooses a random number \( i \in \{1,2\} \) and challenges the server to identify the \( estub^d \) of \( subtree(n_i) \).

- If the server can now guess the value \( i \) with Probability greater than \( 1/2 \), then the partitioning strategy followed in the \( estub^d \) is not semantically secure.

Consider the employee schema introduced in fig 6.3. Only the \( Age \) and \( Salary \) attribute are considered and the \( Name \) attribute is ignored for simplicity. Fig 6.1 illustrates the single dimensional partitioning of the \( Age \) and salary attributes. An Egui-depth histogram is constructed on both the attributes. The black points represents the plaintext values of \( Age \) and \( Salary \) nodes for 10 different XML documents. The server is unaware of the plaintext values, but is aware of the partition identifiers stored. Let us say that now, the server chooses two XML documents with \( Age = 72 \)
Figure 6.2: Multidimensional partitioning

Figure 6.3: Employee Schema
years, salary = 8K and Age = 40 and salary=65k. These two points have an arrow pointing to them in the fig 10. The client now returns <P(Age)= 42, P(Salary)= 3> and <P(Age) = 21 and P(salary)= 34> and challenges the server to find out the partitions belonging to 72 year old who earns 8K per year. The server can now do a frequency analysis and and find out that there are two documents with <P(Age) = 21 and P(salary)= 34> and none with <P(Age)= 42, P(Salary)= 3 >. Using domain knowledge that 72 years olds do not usually earn 8K per year, the server now concludes that <P(Age)= 42, P(Salary)= 3 > belongs to document with Age = 72 and Salary= 8k, with a very high probability. A formal proof for the lack of semantic security for the single dimensional partitioning is given in [18]. Now consider the same database represented using the multidimensional partitioning strategy illustrated in Fig 6.2. A multidimensional equi-depth histogram is constructed on multidimensional domain[23]. For the same XML documents the client now returns partitions 15 and 92. The server now cannot distinguish between the two partitions as they have the same frequency and cannot come to a meaningful conclusion. This indistinguishability makes the equi-depth multidimensional histogram secure against inferences that are drawn using the domain knowledge of the data. Semantic security is a strong notion of security and a data owner should choose the equidepth- multidimensional partitioning strategy to withstand against such attacks.

**K-anonymity:** K-anonymity was introduced as a measure of privacy in [19]. K-anonymity model was proposed for scenarios where private data released of subjects does not allow the adversary to identify the subjects. K-anonymity model is useful to analyze the known plaintext attack. In the Known Plaintext attack, the server
has one or more \( \text{subtree}(n), \text{estub}^d \), where \( d \in \{s, m\} \) pairs. K-anonymity is only preserved if for every \( \text{estub}^d \) in \( \text{subtree}(n), \text{estub}^d \), there are at least \((K-1)\) other \( \text{estub}^d \)s with the same partitioning information. Such a guarantee cannot be given in a single dimensional partitioning strategy, since dimensions are partitioned individually. Whereas in the multi dimensional partitioning strategy if care is taken that every partition has at least \( K \) estubs mapping to it, then K-anonymity is preserved.
Chapter 7

Query Translation

This section explores how a client query $Q$ can be transformed into a set of queries $Q^S = \{ Q^S_1, Q^S_2 \ldots Q^S_N \}$, that can be executed at the server side over the encrypted data representation. The results of all the queries in $Q^S$ will the decrypted and filtered at the client side to compute the actual answer. Our objective is to push the majority of the work to the server side.

The XML query model developed in [10] is used to model XML queries as pattern trees. Pattern trees are pairs $P = (T, F)$ where $T$ is a node labeled tree and $F$ is a boolean combinations of predicates on nodes that belong to $T$. Fig 7.1 shows the example of a pattern tree, that corresponds to a query $Q$ seeking the content of Name in an XML document where the email and title of the paper has been specified as “I.M.Author@email.com” and Title = “Database stuff” respectively. This is an example query on the XML database conforming to the schema in Fig 2.1. Every edge of the pattern tree is either labeled as PC(parent-child) and AD(Ancestor-dependency), which describes the relationship between the nodes. For the rest of the thesis, XML
queries are viewed as pattern trees. Given XML query $Q$ and its pattern tree representation ($Q.T$, $Q.F$), our objective is now is to map the query to a set of server-side queries $Q^S = \{ < Q_1^S.T, Q_1^S.F >, < Q_2^S.T, Q_2^S.F >, \ldots < Q_N^S.T, Q_N^S.F > \}$, such that queries in $Q^S$ can be evaluated over the encrypted XML representation.

Intuitively, the security primitives can be re-applied on $Q$ to get the required set of queries to be executed at the server side. Fig 7.2 describes our overall query translation strategy. In step 1, the implicit structure hidden in $Q.T$ is unraveled. The edges with AD relationship in $Q.T$ are resolved into a path of PC edges using
the original XML schema of the unencrypted documents. This is done to uncover potential nodes hidden in the AD edge that could have Encryption primitives specified on them. There could be leaf nodes in $Q.T$ which do not correspond to leaf nodes in the schema of the unencrypted documents. For instance, the node $Name$ in Fig 7.2, is not a leaf node in the XML schema in Fig 2.1. The $subtree(Name)$ from the original XML schema is now placed under the node $Name$ in $Q.T$, to uncover other encryption primitives specified on them. Fig 7.3 illustrates the pattern tree resolved from the initial pattern tree in Fig 7.1. In step 2, the encryption primitives are reapplied on the resolved pattern tree from step 1 to get a set of pattern trees which have the same tree structure as the encrypted XML documents. This is explained in more detail in Section 7.1. In step 3, every predicate in the pattern trees derived from step 2 is mapped to the corresponding server side conditions, which is explained in Section 7.1.

7.1 Mapping the tree structure of the pattern tree

This section describes the individual affect of $E_s$, $E_v$ and $E_t$ primitives on $Q.T'$ (See Fig 7.1). An $E_s(parentNode, childNode)$ will impact $Q.T'$ only if both the
parentNode and childNode belong to it. If that is the case, \( Q \) is split into two pattern trees \( Q_1 \) and \( Q_2 \), similar to the effect of the \( E_S \) primitive on XML schema described in chapter 4. Node \( id \) is introduced as the child of the node parentNode and nodes parentid and Ancillary(parentid) are introduced as the children of the node childNode. Recall that content of the node Ancillary(parentid) is a function of the node \( id \), as during encryption of the XML document, the content of the \( id \) node is partitioned and partition identifier was stored as the content of Ancillary(parentid). Clearly, at the server side, the server should first execute the pattern tree which contains the \( id \) node, partition the contents of all \( id \) nodes in the result set and then do a selection on the node Ancillary(parentid) for these partitions. The Formula \( Q.F \) also needs to be split into \( Q_1.F \) and \( Q_2.F \), but this is trivial as the pattern tree can only have predicates in its formula on the nodes which are present in its tree structure. For Example, the effect of \( E_S(paper, author) \) on the pattern in fig 7.3 is shown in fig 7.4. The predicate Ancillary(parentid) = Part(\( Q_1.id \)) in Fig 7.4 is now explained. Function part(\( Q_1.id \)) returns the set of partitions to which the content of the \( id \) node fetched from the execution of \( Q_1 \) at the server. These set of partitions are placed as a predicate on the Ancillary(parentid) node. The \( E_S \) primitive is primarily responsible for splitting the original Query \( Q \) into a set of server side queries.

The \( E_V(n) \) primitive will effect \( Q.T' \) if the node \( n \) belongs to \( Q.T' \). If that is the case the subtree(\( n \)) in \( Q_T' \) is replaced by \( estub^n \). Fig 7.5 shows the effect of the \( E_V(Author) \) primitive on the pattern tree in 7.3.

The \( E_T(node) \) primitive will also encrypt all the nodes in subtree(\( n \)) in \( Q.T \), using the same encryption keys used during the enforcement of the security primitives.
Figure 7.4: Effect of the $E_s$ primitive on the pattern

Figure 7.5: Effect of the $E_v$ primitive on the pattern
7.2 Mapping predicates

Previously, it was shown how query $Q$ represented as $< Q.T, Q.F >$ is split up into a set of pattern trees \( \{ < Q_1.T, Q_1.F >, < Q_2.T, Q_2.F >, \ldots < Q_N.T, Q_N.F > \} \). This section deals with the translation of the predicates on each of these pattern trees to their corresponding server side predicates. Each predicate in $Q_i.F$, where $i \in \{1, 2, \ldots N\}$, is mapped individually to the server side. The predicates specified on the leaf nodes are typically mapped to predicates on the ancillary information. The predicate could be any one of the following 4 types a) $\text{leafNode.content} = \text{Value}$ b) $\text{leafNode.content} < \text{Value}$ c) $\text{leafNode_k.content} = \text{leafNode_l.content}$, where $k \neq l$ d) $\text{leafNode_k.content} > \text{leafNode_l.content}$, where $k \neq l$. Only one of these cases is discussed in the interest of brevity. It is hoped that this gives the reader enough intuition to understand the condition mapping procedure.

$\text{leafNode.content} < \text{Value}$ If the $\text{leafNode}$ is not encrypted when it was stored at the server, there isn’t any requirement for condition mapping. Only if the $\text{leafNode} \in \text{estub}^m$ there is a necessity to transform this condition to the server side condition. If the $\text{leafNode} \in \text{estub}^m$, and if the path from the node where the $E_V$ primitive was specified to the $\text{leafNode}$ does not contain a multi-valued operator, then the $\text{leafNode}$ participated in the multi-dimensional partitioning. All the multi-dimensional partitions which contain points less than Value in the $\text{leafNode}$ dimension need to be fetched from the server side. The condition now is transformed onto the $\text{Ancillary}^1(L_1 \ldots L_N)$ node. For instance consider the Multi-dimensional partitioning of the $\text{subtree(Author)}$ domain introduced in Table 5.1. For the condition $id < 36$, partitions 28,99 and 5000 have at least one value less than 36 in the $id$ dimension. The server side condition now is $\text{Ancillary}^1(L_1 \ldots L_N) \in \{28,99,5000\}$. If the path from the node on
which the $E_V$ primitive to the $\text{leafNode}$ contains a multi-valued operator, then the $\text{leafNode}$ was single dimensionally partitioned. A LIKE query is executed on the $\text{Ancillary}^2(L_1 \ldots L_N)$ node for all the partitions in the $\text{leafNode}$ dimension, which contain points less than $Value$. For example if the path from node where the $E_V$ primitive was specified to $\text{leafNode}$ contained a "*" operator, the conditions $id < 36$ is mapped to $\text{Ancillary}^2(L_1 \ldots L_N) \in \{^{*28*},^{*99*},^{*5000*}\}$. 
Chapter 8

Experiments

Due to the security mechanisms developed in this thesis, there is extra cost paid by the client for query processing. This extra cost includes the encryption, decryption of data and also the network cost for transferring the superset of results. In this chapter this extra cost is analyzed and the proposed techniques validated.

**Dataset and Queries:** An XML document generator is implemented which generates XML documents confirming to partial Xbench [15] schema, which is shown in Fig 8.1. The query set used in experimentation is shown in Table 8.1.

**Setup Details:** The experiments were conducted using a 4 ultra sparc processors and 3GB RAM machine as the server and the Intel pentium 3 based machine with 512MB RAM as the client. The XML documents generated were mapped to a relational database using the technique in [11]. IBM DB2 V7.1 was used to run our queries at the server. Blowfish encryption algorithm[20] was used to encrypt the data when stored at the server side. An equi-width histogram is used to partition the data stored at the server.
Figure 8.1: Schema of the XML documents used in the experiments

Q1: For $order in input()//order
   where order/credit_card_transaction/amount gt 9991 return
   <name>
   {order/credit_card_transaction/name}
   </name>

Q2: For $order in input()//order
   where order/order_lines/orderline/discount = 10
   return <order_id> {$order/@id} <
   /order_id>

Table 8.1: Query set used in the experiments

<table>
<thead>
<tr>
<th>Size of the Database</th>
<th>$stub\textsuperscript{2}</th>
<th>$stub\textsuperscript{3}</th>
</tr>
</thead>
<tbody>
<tr>
<td>10 mb</td>
<td>106 msec</td>
<td>143 msec</td>
</tr>
<tr>
<td>20 mb</td>
<td>225 msec</td>
<td>289 msec</td>
</tr>
<tr>
<td>50 mb</td>
<td>275 msec</td>
<td>367 msec</td>
</tr>
<tr>
<td>100 mb</td>
<td>1296 msec</td>
<td>1421 msec</td>
</tr>
</tbody>
</table>

Table 8.2: Client side performance of the $stubs for Query $Q_1$
There are two different experiments that were conducted. The first experiment measures the relative performance of $estub^e$, $estub^m$ and $estub^1_m$. $estub^1_m$ has similar structure to $estub^m$, but instead of storing the encryption concatenation of the leaf nodes (i.e. $E(L_1 \ldots L_N)$), $estub^1_m$ stores individual encrypted values for all leaf nodes. Therefore, the required encrypted content can be fetched from the server, instead of the total encrypted string. From the XML documents that were generated, the $\text{subtree}(\text{Credit\_Card\_Transaction})$ was replaced with each of the above structures. Since $\text{subtree}(\text{Credit\_Card\_Transaction})$ does not have a “*” or a “+” operator, node $\text{Ancillary}(L_1 \ldots L_N)$ was not maintained for both $estub^m$ and $estub^1_m$. For every leaf node $L_i$, $K_i$ number of partitions maintained in $estub^e$. $\prod_{i=1}^{d} K_i$ number of multi-dimensional partitions were maintained for $estub^m$ and $estub^1_m$, where $d$ is the number of dimensions. This was done to be fair for both the partitioning strategies. Because of the bigger domain space multi dimensional partitioning strategy is more likely to contain more partitions than any of the single dimensional partitioning strategies of the leaf nodes. An Index was maintained on partition information on all the leaf nodes in the case of $estub^e$ and on the multi dimensional partition information for $estub^m$ and $estub^1_m$ to speedup the query execution. Query $Q_1$ was first translated into a query on the encrypted data using the techniques in chapter 5 and then translated into an SQL query at the server. Fig 8.3 shows the server side performance of all the structures. $estub^e$ performs better than $estub^m$ and $estub^1_m$, who perform relatively the same. Due to the larger size of the index of $estub^m$ and $estub^1_m$, they perform worse than $estub^e$. The plot Uncrypted illustrates the time taken in a model where the data was stored unencrypted at the server side. Clearly, all the structures perform worse than the unencrypted plot. Since the range of the query $Q_1$ gets mapped to
random partitions, multiple index access was necessary at the server side, hence the performance degradation.

The relative performance of the structures were also measured at the client. The client side cost includes decryption, filtering and XML reconstruction. For estub$m$ it also includes parsing the content of the individual nodes from the string concatenation. estub$m$ fetches the encrypted value of the string concatenation of the content of all the leaf nodes and ends up paying the extra cost due to transmission. Encryption algorithms have a high startup cost and it is always profitable to do bulk decryption and therefore estub$m$ pays lower decryption cost than estub$_i^m$, which might start the decryption process multiple number of times. If the data owner wants to retrieve the majority of the leaf nodes, then he could use estub$m$ instead of estub$_i^m$. For $Q_1$ however, since estub$_i^m$ fetches only one encrypted node, it fared better than estub$m$ as shown in Table 8.2. estub$_i$ has the same client side performance to that of estub$m$ and hence not shown in Table 8.2.

In the second experiment the subtree(order_lines) from the generated XML documents was replaced with estub$m$ and stored them at the server. Since subtree(order_lines) contained a “+” operator, Node Ancillary$(L_1 \ldots L_N)$ was also used for storing ancillary information, unlike our earlier experiment. Performance of estub$m$ was compared with the unencrypted model, since estub$_i$ cannot handle this case. Query $Q_2$, after undergoing the required transformation, was executed at the server.
Figure 8.2: Server side performance for $Q_2$

Fig 8.2 shows that $estub^m$ performs worse, as the $Q_2$ is mapped as a LIKE query on node $Ancillary^2(L_1 \ldots L_N)$ and LIKE queries are evaluated as a table scan on many commercial RDBMS.
Figure 8.3: Server side performance for $Q_1$
Chapter 9

Related Work

Previous work in XML security has been addressed in the context of XML access control [6][7][9]. Access control models allow authenticated users to access information, if they have the required privileges. Types of access explored in the literature include read, write, execute, update, query . . . etc. In the context of XML, authenticated users are allowed to read or modify a partial XML document. The authorizations/privileges are granted at the node (an element or an attribute) level, which is the granularity that XML offers.

In [7] the authors present a language for specifications of access restrictions, both at the instance-level and the schema-level for XML documents. In [9], the authors present provisional access to XML documents for enforcing access control. Previous XML access models discussed above, issued an yes/no response to user’s access requests. In provisional access the users is allowed access a certain subset of the XML document, based on the knowledge that users possesses. This model is useful in many real-life e-commerce applications. All of the above were based on a trusted
server model. The data belonged to the server, and the server assigned or denied privileges to users. In the DAS model explored in this paper, data actually belongs to the client and the server is untrusted. Untrusted server model for XML access control was explored in Mikalu et al [8] and E.Bertino et al [5]. They propose cryptographically enforcing access control on published XML documents. The data owner publishes his information after encrypting the parts of the XML documents to enforce his access policy, which are downloaded by the clients. Clients can do the query processing after decrypting the XML documents, provided they have the knowledge of the encryption keys. It is assumed that data owner distributes the keys to the clients through some secure channel. Our work deals with query processing on the encrypted XML documents which has not been addressed before.

Hacigumus et al [2] propose techniques to execute SQL queries over encrypted relational databases. The techniques proposed in this work cannot be applied to XML data because of its rich structure and presence of set based elements. Motivated by strategy in [2] we store ancillary information along with encrypted data to facilitate query processing. However we propose a different strategy multi-dimensional partitioning which overcomes some of the security limitations of the previous work.

The authors in [28] propose techniques to execute aggregation queries over encrypted relational databases which is an extension to their earlier work in [2]. The main idea is use to privacy homomorphisms encryption schemes which allow encrypt data to be operated on without the knowledge of the decrypted data.
This thesis does not deal with aggregation queries. We suspect similar techniques could be applied to XML data as well.

The authors in [25] have proposed techniques of optimal partitioning and diffusion based repartitioning of single dimensional domains. These techniques can also be applied to the multi-dimensional partitioning strategies discussed in this paper.

The authors in [30] explore a variation of a DAS model where the service provider is partially trusted. The service provider can have access to the plaintext data, but is its responsibility to protect unauthorized access to the client’s data. Sensitive data is stored on the disk in encrypted form, whenever a data is fetched from the disk it is decrypted in the main memory which is assumed to be secure. They propose optimized storage models for relational data that reduces encryption/decryption time when the data is fetched or stored. This thesis explores a model where the service provider is fully untrusted and hence differs from this work.

The authors in [31] explore the same DAS model and provide a hash based solution for generating ancillary information. They also present a quantitative evaluation of the security provided by their methods. This work only assumes selection queries and does not work with range queries. This thesis proposes techniques which also work with range queries.
Chapter 10

Conclusions

This thesis presented techniques to support query processing on encrypted XML data in outsourced database model. Such a problem predominantly occurs in “Database as a Service” (DAS) architectures, where a client may outsource data to a service provider that provides data management service. Previous methods suggested for executing queries over encrypted relational data cannot be applied for XML data due to the rich structure it posses, and the presence of the set based elements. Our techniques, inspired by the previous methods, also store ancillary information along with the encrypted data, although we differ in the type of ancillary information stored. A novel multidimensional partitioning strategy was introduced, which addresses the limitations of the previous strategies proposed. Most of the work for query processing was done at untrusted server side on encrypted data. Ancillary information was used to prune the search space during query processing. A superset of answers are returned, which are later decrypted at the clients side and unwanted data filtered out. Security provided by the proposed techniques was then investigated against some known cryptographic attack models. Experiments were done to validate the
proposed techniques. Some of the techniques proposed here could also be used in the context of relational databases, but such a study is out of the scope of this thesis.

There are many open problems that need to be addressed in the future. This thesis deals with only Selection-projection-join queries. Execution of aggregation and LIKE queries was not addressed here. Recently the authors in [28] proposed techniques to execute aggregation queries over encrypted relational databases. There is need for different kinds on ancillary information/indices to support these queries over encrypted data. It will be interesting to see if the proposed solution can be adapted to the context of XML databases as well.

A formal quantification of the security provided by the multidimensional partitioning strategy needs to be investigated. Hore et al.[25] propose variance and entropy as the measures of security for single dimensional partitioning scheme. We suspect that both these measures could also be used for the case of multi dimensional partitioning scheme.

This thesis does not make any assumption about the storage models for the XML documents. XML documents are primarily stored using either a native XML database or a relational database engine. Techniques proposed in this thesis work with both the models. Since Encryption and Decryption are CPU intensive processes there is a need for optimized storage models which reduce encryption/decryption time. This is an open problem and needs to be explored.
This thesis does not deal with query optimization techniques in the DAS model. Existing optimization techniques are not applicable in this model due to the presence of Encryption/decryption operations and transmission of data over the network which significantly changes the cost model assumed in the traditional model.


Bibliography


[25] Bijit Hore, Sharad Mehrotra, Gene Tsudik. Privacy/Preserving Index for Range queries. *In the Proc of 2004 Very Large Databases (VLDB).*


