Secure Query Answering and Privacy-preserving Data Publishing

by

Hui Wang

B.Sc., Wuhan University, P.R.China, 1998
M.Sc., The University of British Columbia, 2002

A THESIS SUBMITTED IN PARTIAL FULFILMENT OF
THE REQUIREMENTS FOR THE DEGREE OF
Doctor of Philosophy

in

The Faculty of Graduate Studies
(Computer Science)

The University Of British Columbia
October, 2007
© Hui Wang 2007
Abstract

The last several decades have witnessed a phenomenal growth in the networking infrastructure connecting computers all over the world. The Web has now become an ubiquitous channel for information sharing and dissemination. More and more data is being exchanged and published on the Web. This growth has created a whole new set of research challenges, while giving a new spin to some existing ones. For example, XML (eXtensible Markup Language), a self-describing and semi-structured data format, has emerged as the standard for representing and exchanging data between applications across the Web. An important issue of data publishing is the protection of sensitive and private information. However, security/privacy-enhancing techniques bring disadvantages: security-enhancing techniques may incur overhead for query answering, while privacy-enhancing techniques may ruin data utility. In this thesis, we study how to overcome such overhead. Specifically, we address the following two problems in this thesis: (a) efficient and secure query evaluation over published XML databases, and (b) publishing relational databases while protecting privacy and preserving utility.

The first part of this thesis focuses on efficiency and security issues of query evaluation over XML databases. To protect sensitive information in the published database, security policies must be defined and enforced, which will result in unavoidable overhead. Due to the security overhead and the complex structure of XML databases, query evaluation may become inefficient. In this thesis, we study how to securely and efficiently evaluate queries over XML databases. First, we consider the access-controlled database. We focus on a security model by which every XML element either is locally assigned a security level or inherits the security level from one of its ancestors. Security checks in this model can cause considerable overhead for query evaluation. We investigate how to reduce the security overhead by analyzing the subtle interactions between inheritance of security levels and the structure of the XML database. We design a sound and complete set of rules and develop efficient, polynomial-time algorithms for optimizing security checks on queries. Second, we consider encrypted XML database in a "database-as-service" model, in which the private database is hosted by an untrusted server. Since the untrusted server has no decryption key, its power of query processing is very limited, which results in inefficient query evaluation. We study how to support secure and efficient query evaluation in this model. We design the metadata that will be hosted on the server side with the encrypted database. We show that the presence of the metadata not only facilitates query processing but also guarantees data security. We prove
that by observing a series of queries from the client and responses by itself, the server's knowledge about the sensitive information in the database is always below a given security threshold.

The second part of this thesis studies the problem of preserving both privacy and the utility when publishing relational databases. To preserve utility, the published data will not be perturbed. Instead, the base table in the original database will be decomposed into several view tables. First, we define a general framework to measure the likelihood of privacy breach of a published view. We propose two attack models, unrestricted and restricted models, and derive formulas to quantify the privacy breach for each model. Second, we take utility into consideration. Specifically, we study the problem of how to design the scheme of published views, so that data privacy is protected while maximum utility is guaranteed. Given a database and its scheme, there are exponentially many candidates for published views that satisfy both privacy and utility constraints. We prove that finding the globally optimal safe and faithful view, i.e., the view that does not violate any privacy constraints and provides the maximum utility, is NP-hard. We propose the locally optimal safe and faithful view as the heuristic, and show how we can efficiently find a locally optimal safe and faithful view in polynomial time.
# Contents

Abstract .................................................. ii

Contents .................................................. iv

List of Figures ........................................ viii

Acknowledgements ..................................... xi

Dedication ............................................. xiii

## 1 Introduction ........................................ 1
  1.1 Efficient and Secure Evaluation of Queries over XML Database ..... 1
      1.1.1 Efficient and Secure Evaluation of Queries over Access-Controlled XML Databases ... 2
      1.1.2 Efficient and Secure Evaluation of Queries over Encrypted XML Databases .......... 4
  1.2 Privacy-and-utility-preserving Data Publishing ................. 6

## 2 Background .......................................... 9
  2.1 eXtensible Markup Language (XML) .......................... 9
      2.1.1 Document Type Definition (DTD) .............. 11
      2.1.2 XML Tree Pattern Queries (TPQ) ............. 13
      2.1.3 Structural Join: Efficient Evaluation of XML Queries ....... 14
      2.1.4 XPath Query Containment .................. 15
  2.2 XML Encryption .................................... 16
  2.3 K-anonymity ...................................... 16

## 3 Query Evaluation over Access-controlled XML Databases ....... 20
  3.1 Related Work ...................................... 22
      3.1.1 XML Access-Control Models .................. 22
      3.1.2 Efficient Enforcement Mechanisms .......... 24
      3.1.2.1 Authorization Views .................. 24
      3.1.2.2 Access-control Index .............. 25
      3.1.2.3 Static Analysis .................. 26
      3.1.2.4 Query Evaluation and Rewriting ...... 26
  3.2 Model and Problem ................................ 27
      3.2.1 Security and Authorization Model .......... 27
3.2.2 Multi-level Security and Authorization ........................................... 28
3.2.3 SC Annotations and Problem Statement ........................................... 29
3.3 Definitions and Conventions ............................................................... 30
3.4 DAG-structured DTD Graphs ............................................................... 31
  3.4.1 Root Rule ....................................................................................... 32
  3.4.2 Parent-Child(PC) Rule ................................................................. 33
  3.4.3 Child-Parent(CP) Rule ................................................................. 34
  3.4.4 Cousin-cousin(CC) Rule ............................................................... 34
    3.4.4.1 Unconstrained Nodes ....................................................... 35
    3.4.4.2 Constrained Nodes .............................................................. 36
  3.4.5 Putting It All Together ................................................................. 40
3.5 DAG-Structured DTDs with Disjunction ............................................. 43
  3.5.1 The Challenges ............................................................................ 43
  3.5.2 Optimizing against Disjunction DTDs ......................................... 45
  3.5.3 Modified Rules ........................................................................... 46
  3.5.4 Algorithm DisjunctionOpt ......................................................... 48
  3.5.5 Theoretical Results of Disjunction DTDs ...................................... 49
3.6 Experiments ......................................................................................... 49
  3.6.1 Experimental Setup ..................................................................... 50
  3.6.2 Impact of Optimization .............................................................. 50
  3.6.3 Optimization Overhead .............................................................. 52
3.7 Summary ............................................................................................. 53
4 Query Evaluation over Encryption XML Databases ................................ 56
  4.1 Related Work ................................................................................... 59
  4.2 Security Issues .................................................................................. 60
    4.2.1 Encryption Scheme ................................................................... 60
    4.2.2 Security Constraints ................................................................ 61
    4.2.3 Prior Knowledge and Attack Model ....................................... 62
    4.2.4 Security Definitions ................................................................. 63
  4.3 Encryption Scheme ........................................................................... 66
    4.3.1 Secure Encryption Scheme ...................................................... 66
    4.3.2 Optimal Secure Encryption Scheme ....................................... 69
  4.4 Metadata on Server ....................................................................... 70
    4.4.1 Structural Index ........................................................................ 70
    4.4.2 Representing DSI index ........................................................... 72
    4.4.3 Value Index .............................................................................. 76
      4.4.3.1 Order-preserving Encryption with Splitting, Scaling and Noise ...................................................................... 77
  4.5 Query Processing .............................................................................. 84
    4.5.1 Query Translation at the Client ............................................... 84
    4.5.2 Query Translation at the Server ............................................... 85
    4.5.3 Security of Query Answering ................................................... 86
    4.5.4 Query Post-Processing ............................................................. 90
  4.6 Activities on Client Side .................................................................. 91
    4.6.1 Key Management ...................................................................... 91
4.6.2 Client’s Benefit ........................................ 92
4.7 Experimental Evaluation .................................. 92
  4.7.1 Experimental Setup .................................. 93
  4.7.2 Our Approach vs. Naive Method ..................... 94
  4.7.3 Division of Work between Client and Server .......... 94
  4.7.4 Effect of Query Size ................................ 95
  4.7.5 Effect of Various Encryption Schemes ................ 95
4.8 Summary .................................................. 96

5 Probabilistic Analysis of Privacy Leakage of Published Views 100
  5.1 Related Work ........................................... 103
  5.2 Security Model .......................................... 104
    5.2.1 Private Association ............................... 104
    5.2.2 Attack Models ....................................... 104
      5.2.2.1 Unrestricted Attack Model ....................... 104
      5.2.2.2 Restricted Attack Model ......................... 105
      5.2.2.3 Probability of Privacy Breach ................. 106
  5.3 Connectivity Graph .................................... 106
    5.3.1 Two-View-Table Case .............................. 107
    5.3.2 K-View-Table Case ................................ 108
  5.4 Measurement of Probability of Privacy Breach .......... 108
    5.4.1 Two-View-Table Case .............................. 109
    5.4.2 K-view-table Case ................................ 113
  5.5 Summary ................................................ 114

6 Privacy and Utility Preserving Data Publishing by Views .... 115
  6.1 Related Work .......................................... 116
  6.2 Security Model .......................................... 117
    6.2.1 Private and Public Associations .................... 117
    6.2.2 Association Cover, Privacy, and Utility .......... 119
  6.3 Faithful View Schemes ................................ 121
  6.4 Connectivity Graph .................................... 122
    6.4.1 The Concept ....................................... 122
    6.4.2 Traversing Connectivity Graph ..................... 125
  6.5 View Lattice .......................................... 126
    6.5.1 The Concept ....................................... 126
    6.5.2 Properties of View Lattice ......................... 129
      6.5.2.1 Monotonicity .................................. 129
      6.5.2.2 Equivalence .................................. 130
    6.5.3 Optimization of Search Strategy .................. 133
  6.6 Experiments ............................................ 135
    6.6.1 Experimental Setup ................................ 135
    6.6.2 Size of Connectivity Graph ......................... 136
    6.6.3 Locally Optimal Solution vs. Globally Optimal Solution 137
    6.6.4 Performance ....................................... 137
    6.6.5 Optimization ...................................... 138
## Contents

6.7 Summary ................................................. 139

7 Conclusions and Future Work .................................. 140
  7.1 Future Work ........................................... 141
    7.1.1 Efficient and Secure Query Evaluation .............. 141
    7.1.2 Privacy-preserving Data Publishing ................. 141

Bibliography .................................................. 143

A Proof ....................................................... 152
# List of Figures

1.1 An Example of Encrypted XML Database .............................................. 2  
1.2 Example XMark-based DAG DTD .......................................................... 2  
1.3 Base Table T ......................................................................................... 6  
1.4 Example of Unsafe Views ....................................................................... 6  
1.5 Example of Safe Views .......................................................................... 7  
2.1 An XML Document of Student Record .................................................. 10  
2.2 An Example of an XML Tree .................................................................. 10  
2.3 the DTD of Student Record .................................................................... 11  
2.4 DTD Graph illustrated: (a) (Original) DTD graph and (b) simplified DTD graph of \{A\rightarrow(B?C^*)? | (CD^*), B\rightarrow(EF^+)\} ................................................................. 13  
2.5 Example: DAG of XMark DTD ............................................................... 13  
2.6 Examples of Tree Pattern Query ......................................................... 14  
2.7 Examples of 3-tuple Index for Structural Join ....................................... 15  
2.8 Various Encryption Schemes to Protect Credit Card Number ............... 17  
2.9 Example of K-anonymity ....................................................................... 18  
3.1 Example XMark-based DAG DTD ......................................................... 21  
3.2 Example XMark-based DAG DTD: nodes not marked M (mandatory) or O (optional) do not have their security explicitly defined. .................................................. 28  
3.3 Examples for DAG-Structured DTD: PC Rules .................................... 32  
3.4 Root Rule & Optimization ................................................................... 32  
3.5 Parent-Child Rule ................................................................................ 33  
3.6 CC Rule Example for Unconstrained Nodes ........................................ 35  
3.7 CC Rule for Unconstrained Nodes: (a) Case IV.1; (b) Case IV.2; (c) Case IV.3 ............................................................................................................ 36  
3.8 CC Rule for Unconstrained Nodes ....................................................... 37  
3.9 CC Rule Examples for Constrained Nodes ........................................... 38  
3.10 CC Rule for Constrained Nodes: (a) Case V.1; (b) Case V.2; (c) Case V.3 ............................................................................................................ 38  
3.11 CC Rule for Constrained Nodes ........................................................... 39  
3.12 Example of Candidate DTDs ............................................................... 43  
3.13 A disjunction DTD ............................................................................... 44  
3.14 An Example Database Instance; Nodes marked \times (\checkmark) are inaccessible (accessible). .................................................................................. 44  
3.15 A disjunction DTD and a query ............................................................ 46
List of Figures

3.16 Queries Used in the Experiments ........................................ 50
3.17 DTDs w.r.t setup of overriding nodes ............................... 51
3.18 Optimized SC annotation on query .................................... 52
3.19 Query Evaluation Time on Q1, Q2 and Q4 ......................... 53
3.20 Query Evaluation Time on Various Densities ....................... 54
3.21 Optimization Overhead w.r.t Size of Query ......................... 54
3.22 Query Optimization Overhead with Regard to Number of Dis­
junction Nodes ................................................................. 55

4.1 System Architecture .......................................................... 58
4.2 An Encrypted XML Health Care Database ............................ 61
4.3 Calculating DSI Index ....................................................... 71
4.4 Insert node P between p1[min1, max1] and p2[min2, max2]. Cal­
culate P's interval [min, max] ............................................. 72
4.5 Insecurity of Continuous Interval Index ............................... 73
4.6 Structural Index ............................................................... 74
4.7 Possible Structure of Node A .............................................. 74
4.8 Data Distribution before Encryption & after Encryption. X axis: Dis­
tinct (plaintext/ciphertext) Values. Y axis: Number of Occurrence ... 77
4.9 Query Translation on Client ................................................. 84
4.10 Translation of Value-based Constraints ............................... 85
4.11 Constraint Graphs .......................................................... 93
4.12 Saving Ratio by Our Approach, XMark, 10MB ..................... 94
4.13 Query Performance of Various Encryption Schemes, NASA Database,
25MB ........................................................................... 97
4.14 Query Performance of Various Encryption Schemes, XMark Database, 
10MB ........................................................................... 98
4.15 Encryption Times and Sizes regarding Various Encryption Schemes 98
4.16 App and Opt Encryption Schemes vs Top and Sub Encryption Schemes 99

5.1 Example of K-anonymity ...................................................... 101
5.2 Example of Unsafe Views .................................................... 101
5.3 Example of Safe Views ...................................................... 101
5.4 Notations used in Chapters 5 and 6 ..................................... 103
5.5 An Example of Possible Worlds .......................................... 105
5.6 Probability of Privacy Breach of Two Attack Models ............. 106
5.7 Connectivity Graph of the Two View Tables in Figure 5.3 ....... 107
5.8 Connectivity Graph of Four Views ...................................... 109
5.9 Covers vs. Possible Worlds ............................................... 110
5.10 Restricted Interesting Worlds vs. Unrestricted Possible Worlds 113
5.11 Probability Models .......................................................... 113

6.1 Base Table ..................................................................... 115
6.2 Example of Safe Views ...................................................... 119
6.3 Example of Unsafe Views .................................................. 120
6.4 The Example of Connectivity Graph .................................... 123
List of Figures

6.5 Pseudo Code: ACSizes() ........................................ 125
6.6 Part of View Lattice ........................................... 128
6.7 Example: Order of Join Labels ................................. 131
6.8 Functional Dependency ......................................... 131
6.9 Pseudo Code: FindOutputView() .............................. 132
6.10 Search in Lattice ................................................ 134
6.11 Adult Dataset .................................................. 135
6.12 Size of Connectivity Graph ................................. 136
6.13 Different Configurations ........................................ 139
6.14 Optimization ................................................... 139

A.1 Example XMark-based DAG DTD: nodes not marked M (mandatory) or O (optional) do not have their security explicitly defined. 153
A.2 Counter Example: Proof of Lemma 3.4.1 .................. 153
A.3 CC Rule for unconstrained nodes: (a) case IV.1; (b) case IV.2; (c) case IV.3 ........................................... 154
A.4 CC Rule for constrained nodes: (a) case V.1; (b) case V.2; (c) case V.3 .................................................. 154
A.5 Counter Example: Proof of Lemma 3.4.2 .................. 155
Acknowledgements

Many people have helped me during my stay at the University of British Columbia, and in the years leading up to it. To all of them, I owe my thanks.

I owe my sincerest thanks to my supervisor, Laks V.S. Lakshmanan, for his invaluable advice and encouragement over the five years of my Ph.D. research. He is an inexhaustible source of research problems that are both interesting and well-motivated. He guided me with incomparable patience and wisdom, led me through numerous difficult periods in my research, and helped me find the right path when required. He taught me the nature of research, and how fun it can be.

I would like to thank Dr. Rachel Pottinger and Dr. William A. Aiello, who are members of my Ph.D. committee, for their encouragement and suggestions for this thesis. I thank Dr. Rachel Pottinger (again) and Dr. Raymond Ng for their numerous valuable suggestions concerning my research and other problems, especially public presentation and job hunting. I would like to thank Dr. Divesh Srivastava of AT&T Research for his advice on my work; some of the research work appearing in this thesis was done jointly with him. I also would like to thank Dr. Nicolas Bruno of Microsoft Research and Dr. Serge Abiteboul of INRIA, for their mentoring during my summer internships.

The best thing about pursuing my Ph.D. at UBC was the opportunity to do research with extremely bright and energetic students and colleagues. I would like to thank Ganesh Ramesh, Xiaodong Zhou, Shaofeng Bu, Jessica Zheng Zhao, Wookey Lee, and all the others who made research at UBC an extremely enjoyable experience. I do enjoy working with them.

My thanks also go to the graduate and graduated students who are (were) the members of the database group of UBC. In particular, I thank Xiaodong Zhou, Shaofeng Bu, Timothy Chan, Zhimin Chen, Maryann Yan Zhao, Jessica Zheng Zhao, Elaine Qing Chang, Xun Sun, Betrand Low, Terence Ho, Jie Zhao, Shuan Wang, Andrew Carbonetto, and Michael Lawrence. They came to my paper presentations and practice talks, discussed my research, read my papers, and gave me numerous suggestions. We had many great times together.

My seven years at UBC have been a wonderful experience, and I thank all of my friends who made it so. I owe my special thanks to Juan Li, who is one of my best friends at UBC. I would also like to thank Carson Leung, Ed Knorr, Ying Su, Xiaohong Zhao, Jian Chen, Shihao Gong, Xiaoming Zhou, Zhenyu Song, Huawen Luo, Ruiyao Yang, Dan Cristian Tulpan, and many others for their friendship and support.

Most of all, I would like to thank my family, my brother Zhi Wang with whom
I have shared many memorable moments, my parents, who always believed in me, and gave me unconditional love, support and encouragement at every step of my life, and my husband Chao Li, who shares this world with me with his love and passion. I dedicate this thesis to them.
Dedication

To my families.
Chapter 1

Introduction

The last several decades have witnessed a phenomenal growth in the networking infrastructure connecting computers all over the world. The Web has now become an ubiquitous channel for information sharing and dissemination. This growth has created a whole new set of research challenges, while giving a new spin to some existing ones. For example, XML (eXtensible Markup Language), a self-describing and semi-structured data format, has emerged and become a standard to represent and exchange data between applications across the Web. An important issue for information sharing and publishing is the protection of the sensitive and private information. According to the "computer crime and security survey" published by the Computer Security Institute (CSI) and the Federal Bureau of Investigation (FBI) [97], the total financial loss of year 2006 caused by database attacks is over $130 million for 700 companies and organizations involved in the survey. Therefore, security/privacy-enhancing techniques are necessary. However, the security/privacy-enhancing techniques bring disadvantages: security-enhancing techniques may incur overhead for query answering, while privacy-enhancing techniques may ruin data utility. In this thesis, we address these two problems: (1) efficient and secure query evaluation over XML databases, and (2) publishing relational databases while protecting privacy and preserving utility.

1.1 Efficient and Secure Evaluation of Queries over XML Database

The first part of this thesis deals with the efficiency and security issues of query evaluation over XML databases. XML is used as the standard for data integration and exchange on the Internet (more details about XML can be found in Chapter 2). Since XML plays an important role in data representation on the Web, more and more sensitivity is associated with XML data. Therefore, security policies must be defined and enforced on an XML database to protect sensitive data. However, the enforcement of those security policies will result in unavoidable overhead for query evaluation. Example 1.1.1 illustrates more details of such security overhead.

Example 1.1.1 [Security Overhead for Query Evaluation]

Figure 1.1 shows an example XML database. To protect the patient records in the database, three patient elements are encrypted individually in blocks (rep-
Chapter 1. Introduction

Figure 1.1: An Example of Encrypted XML Database

Figure 1.2: Example XMark-based DAG DTD

represented as squares). To answer the query $Q$ that asks for the patient’s record whose insurance policy number is 26544, if there is no index to help locate the answer in the encrypted database, all the three encryption blocks must be decrypted for query answering. However, since only the first encryption block contains the answer of the query, the decryption of the other two encrypted blocks is irrelevant to the query answering, which results in unavoidable overhead.

It is possible that security overhead dramatically degrades the performance of query evaluation. Consequently, we study how to optimize the security overhead of query evaluations on XML databases, so that the queries can be answered efficiently, while the data is still securely protected. We study this problem on both access-controlled XML databases and encrypted XML databases, described below.

1.1.1 Efficient and Secure Evaluation of Queries over Access-Controlled XML Databases

XML is widely used in a variety of applications, and has become a standard for describing and exchanging data across the Internet. As more and more XML documents are stored in XML databases, the security of these databases has become an important issue. Access control is one of the methods used to guarantee the security of XML databases. Recent research on XML access control
has focused on the complementary problems: (1) development of sophisticated access control models (for example, [16, 34, 35]), and (2) efficient mechanisms to enforce policies in these models (for example, [10, 34, 35, 42, 56, 95]). A naive approach to secure XML query evaluation is: (i) compute the query result without consideration of any security policy, and (ii) filter the query result using the access-control policies in a post-processing step. While this approach may appear attractive, it is not secure. Example 1.1.2 illustrates the reason.

Example 1.1.2 [Insecurity of Filtering Query Results]
Consider the XML database with its DTD illustrated in Figure 1.2, which has information about items and customer accounts. Assume that a specific user is allowed access to the item (book and software) information but not to any account information. Thus for any query that is related with account information, the query result returned to the user should be empty. However, if only query results are filtered for accessibility, then the following query: \(/OnlineSeller[. //Account/ID='00190']//Book\), which asks for the (accessible) book information if there exists an account whose ID is 00190, would return non-empty answers if such an account exists. This is clearly not the desired intent of the access-control policy, since it allows the user to check the existence of account 00190.

This example shows that we need to ensure that every data element accessed by the query is authorized to be accessible by the user. In general, by secure query evaluation, user queries only are permitted to check conditions on, and return, XML data that the user is allowed to access.

However, such security checks can cause considerable overhead for query evaluation. Thus it is important to optimize the evaluation of queries over secure databases by minimizing the number of such security checks. In this thesis, we consider the same multi-level security (MLS) model for XML data as in [28], where individual XML elements may each have an explicitly associated security level (represented as an attribute of that element), as do all users. If an element does not have a security level explicitly associated with it, it inherits the security level associated with the closest ancestor where it has a security level explicitly defined. To identify which elements must, may or cannot have security levels explicitly associated with them, the model allows the database schema to be labeled with this information.

Based on the security model, the security checks are implemented by attaching annotations labels to query nodes, which indicate in which way the accessibility of every query node will be determined. However, with a given schema of XML database, as we will see in Chapter 3, the security checks on some query nodes can be accomplished in a more efficient way, or even can be omitted. Thus given the same security model as in [28] and a schema of XML database, we consider the problem of how to optimize the security overhead for query evaluation so that it ensures secure evaluation of the XML databases. The basic idea is that with the security check annotation labels on queries, we rewrite as many of those labels as possible to be cheaper. Compared with [28],...
which focused on much simpler tree-structured schemes for XML databases, we consider more complicated DAG (Directed Acyclic Graph)-structured schemes. Compared with tree-structured schemes, DAG-structured schemes result in exponentially many more schema paths that need to be reasoned. There is an additional source of complexity related to this problem when the schema contains disjunction nodes, which specify the optionality of elements (disjunction nodes are discussed in detail in Chapter 2), since a schema with disjunction nodes is logically equivalent to the disjunction of a potentially exponential number of "candidate" DAG-structured schemas without disjunction. We propose a sound and complete set of rules for optimizing security checks, and develop efficient, polynomial-time algorithms for dealing with the combinatorial explosion. This work was originally published as a chapter in the book "Security in Decentralized Data Management" [88] and is described in Chapter 3.

1.1.2 Efficient and Secure Evaluation of Queries over Encrypted XML Databases

Encryption is a popular technique used to protecting the confidentiality of the information. The W3C (World Wide Web Consortium) has recommended an XML encryption standard [116], which supports different encryption granularities on XML documents (more details concerning XML encryption can be found in Section 2.2). In this thesis, we consider XML encryption in the "database-as-service" (DAS) model [50], in which the data owner, especially those who has a small business and limited budgets, hosts his/her private database on a third-party server. Although the DAS model offers the possibility of reliable storage of large volumes of data, efficient query processing, and savings of database administration costs for the data owner, it raises serious questions about the security of the data managed by the service provider. If the customer is not willing to implicitly trust their data to another party, the privacy of sensitive data must be protected. The same concerns exist for the untrusted disk scenario [66], where the disk is not trusted by the file system over an insecure network. One solution to this problem is that the customer encrypts the sensitive parts of their data, sends the (partially) encrypted database to the service provider, and keeps the decryption key to himself.

Since the server has no decryption key, after the database is encrypted and sent to the server, when the data owner sends queries to the server, the naive approach to query evaluation is that the server returns the whole encrypted database back to the data owner. The cost of data transportation, decryption and query post-processing in this approach is very high. How to efficiently and securely answer the queries over encrypted databases in the DAS model is an important problem. There are several challenges.

The first challenge is the definition of security. What's the meaning of "security" when we state it? How do we measure and determine the security? To address these two issues, we formally define security in the "database-as-service" model, which include the definition of secure encryption scheme, secure metadata, and secure query processing (Sections 4.2.4).
Chapter 1. Introduction

Second, there exist multiple secure encryption schemes to protect the elements in the databases. Those schemes apply encryption at different granularities. For example, encrypting the entire document as an octet sequence, while guaranteeing all sensitive information are protected, quashes any hopes of query optimization: for every query, the server needs to ship the whole encrypted database to the owner, which will decrypt and process the query, at considerable burden to itself. When the granularity is too fine, for example, only leaf elements are encrypted, as we will show in Section 4.3.1, an attacker can infer sensitive information, based on his background knowledge of the domain values and their occurrence frequencies. Thus, the second challenge is to find a secure and optimal encryption scheme (defined in Section 4.3.2), which guarantees that security constraints are enforced while permitting efficient query processing. To address this, we prove that finding an optimal, secure encryption scheme for a given set of security constraints is NP-hard. We propose a mechanism to construct a heuristic secure encryption scheme, which is no worse than twice the cost of the optimal scheme.

Third, after the data is encrypted and sent to the server, in order to enable efficient query processing, some metadata needs to be maintained on the server side. However, careless design of metadata may enable possibilities of information leakage by the untrusted server. Thus the challenge is how to design the metadata so that the hosted database is secure in the presence of the metadata. We propose a specific scheme for the metadata to be maintained by the server, to support efficient query processing. It consists of two key components: the structural metadata called the *discontinuous structural interval index* (DSI) that pertains to the structure of the database, and a B-tree index as the metadata on the values that is based on transforming the unencrypted values by *order-preserving encryption with splitting, scaling and noise* techniques, so that the distribution of unencrypted data is different from that of encrypted data. We prove that in the presence of metadata, the hosted database system is secure.

Finally, by observing the queries and their answers, the attack may infer some knowledge about the sensitive information in the database. We would like to make sure that even after observing a series of queries from the client and responses by itself, the server's knowledge about the sensitive information in the database is always below a given security threshold. By using the metadata, we show how queries can be evaluated efficiently and prove the query answering is secure.

The details of design of metadata, query answering and proofs of security were published in the International Conference on Very Large Database (VLDB) 2006 [89] and are described in Chapter 4.
1.2 Privacy-and-utility-preserving Data Publishing

Many organizations are increasingly publishing data that contains information of individuals. The data will be analyzed to find the hidden properties or patterns of population. We assume the patterns take form of correlations of data values. For example, from the medical database in Figure 1.3, the researchers want to study at which ages people are more likely to have a heart attack. The results, which show the characteristics of population, is vital to society. We call these correlation patterns public associations. In this thesis, we consider the preservation of public associations as utility of data.

Study of public association patterns of a population is important. However, since the published data contains personal information, protecting individual privacy is crucial. We assume that the private information takes the form of associations, which are pairs of values in the same tuple. For example, from the base table in Figure 1.3, neither “Alice” nor “AIDS” alone is a secret, but the fact that they are associated with must be protected. Our goal is to publish the database with private associations (for example, the association between “Alice” and “AIDS”) protected and public associations (for example, the association between age “45” and “heart attack”) preserved.

<table>
<thead>
<tr>
<th>Name</th>
<th>Age</th>
<th>Sex</th>
<th>City</th>
<th>Disease</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>30</td>
<td>F</td>
<td>Toronto</td>
<td>AIDS</td>
</tr>
<tr>
<td>Sarah</td>
<td>30</td>
<td>F</td>
<td>Vancouver</td>
<td>Diarrhea</td>
</tr>
<tr>
<td>Betty</td>
<td>45</td>
<td>F</td>
<td>Vancouver</td>
<td>Heart Attack</td>
</tr>
<tr>
<td>John</td>
<td>30</td>
<td>M</td>
<td>Vancouver</td>
<td>Diarrhea</td>
</tr>
<tr>
<td>Alan</td>
<td>45</td>
<td>M</td>
<td>Vancouver</td>
<td>Heart Attack</td>
</tr>
<tr>
<td>Tim</td>
<td>30</td>
<td>M</td>
<td>Alberta</td>
<td>Diarrhea</td>
</tr>
</tbody>
</table>

Figure 1.3: Base Table T

<table>
<thead>
<tr>
<th>Name</th>
<th>Sex</th>
<th>City</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>F</td>
<td>Toronto</td>
</tr>
<tr>
<td>Sara</td>
<td>F</td>
<td>Vancouver</td>
</tr>
<tr>
<td>Betty</td>
<td>F</td>
<td>Vancouver</td>
</tr>
<tr>
<td>John</td>
<td>M</td>
<td>Vancouver</td>
</tr>
<tr>
<td>Alan</td>
<td>M</td>
<td>Vancouver</td>
</tr>
<tr>
<td>Tim</td>
<td>M</td>
<td>Alberta</td>
</tr>
</tbody>
</table>

\[ v_1 = \Pi_{Name, Sex, City}(T) \]

<table>
<thead>
<tr>
<th>Age</th>
<th>City</th>
<th>Disease</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>Toronto</td>
<td>AIDS</td>
</tr>
<tr>
<td>30</td>
<td>Vancouver</td>
<td>Diarrhea</td>
</tr>
<tr>
<td>45</td>
<td>Vancouver</td>
<td>Heart Attack</td>
</tr>
<tr>
<td>30</td>
<td>Vancouver</td>
<td>Diarrhea</td>
</tr>
<tr>
<td>45</td>
<td>Vancouver</td>
<td>Heart Attack</td>
</tr>
<tr>
<td>30</td>
<td>Alberta</td>
<td>Diarrhea</td>
</tr>
</tbody>
</table>

\[ v_2 = \Pi_{Age, City, Disease}(T) \]

Figure 1.4: Example of Unsafe Views

There are several approaches for privacy-preserving data publishing. These include generalizations [12, 58, 84], cell and tuple suppression [31, 84], adding
Chapter 1. Introduction

<table>
<thead>
<tr>
<th>Name</th>
<th>Age</th>
<th>Sex</th>
<th>City</th>
<th>Age</th>
<th>Disease</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>30</td>
<td>F</td>
<td>Toronto</td>
<td>30</td>
<td>AIDS</td>
</tr>
<tr>
<td>Sarah</td>
<td>30</td>
<td>F</td>
<td>Vancouver</td>
<td>30</td>
<td>Diarrhea</td>
</tr>
<tr>
<td>Betty</td>
<td>45</td>
<td>F</td>
<td>Vancouver</td>
<td>45</td>
<td>Heart Attack</td>
</tr>
<tr>
<td>John</td>
<td>30</td>
<td>M</td>
<td>Vancouver</td>
<td>30</td>
<td>Diarrhea</td>
</tr>
<tr>
<td>Alan</td>
<td>45</td>
<td>M</td>
<td>Vancouver</td>
<td>45</td>
<td>Heart Attack</td>
</tr>
<tr>
<td>Tim</td>
<td>30</td>
<td>M</td>
<td>Alberta</td>
<td>30</td>
<td>Diarrhea</td>
</tr>
</tbody>
</table>

\[ v_1 = \Pi_{Name, Age, Sex, City}(T) \]
\[ v_2 = \Pi_{Age, Disease}(T) \]

Figure 1.5: Example of Safe Views

noise [2, 7, 26], publishing marginals that satisfy a safety range [40], and data swapping [33] (the details of these approaches will be discussed in Section 2.3). Although by these approaches, the privacy of data is protected, since the data is perturbed, as we will see in Chapter 5, the published data may become unusable for its data analysis purpose. Thus we consider a different approach, publishing data by using views. The idea is that the base table will be decomposed into several view tables. No data value will be modified in the views. Therefore, if we put every pair of two attributes whose association must be preserved into the same view table, by using the published views, the same data analysis result is still obtainable. For example, if the base table in Figure 1.3 is split into two view tables \( v_1 \) and \( v_2 \) (as shown in Figure 1.4), since the attributes \( Age \) and \( Disease \) are put in the same table, the result of the analysis of “at which age people are more likely to have a heart attack” is exactly the same as from the base table.

Publishing data by using views solves the utility problem. It may initially seem that the protection of private association is also straightforward: we put any pair of two attributes whose association must be protected into different view tables. However, as pointed out in [94], releasing two attributes in different view tables may still reveal the private information. For example, by joining \( v_1 \) and \( v_2 \) in Figure 1.4, there is only one tuple in the join result associated with \( Alice \). Therefore the attacker still can infer that \( Alice \) has \( AIDS \). However, if the base table is published as shown in Figure 1.5, there will be two diseases, \( AIDS \) and \( diarrhea \), in the join results, that are associated with \( Alice \). Then, based on the join result of the published view tables, the attacker cannot explicitly decide which disease \( Alice \) has. Since different schemes yield different degrees of protection, the problem of designing view schemes is important and challenging.

In this thesis, we study the problem of how to efficiently come up with a view scheme that not only protects all private information but also preserves the maximum utility. First, we study the problem of how to measure the privacy breach of published views by probability. We define two attack models, unrestricted and restricted models. By unrestricted (restricted, respectively) model, we assume the attacker has unrestricted (restricted, respectively) background knowledge. For each attack model, we derive the formulas to quantify the probabilities of privacy breach for the two-view-table case. We also dis-
cuss the difficulty of measuring the probability of a privacy breach when there are more than two view tables available. The details of our attack model and measurement of probability were published in the *workshop on Privacy in the Electronic Society (WPES) 2006* [90], and are described in Chapter 5.

Besides privacy, we also take utility into consideration. We study the problem of how to efficiently decompose the original database and construct published views that are of maximum data utility while maintaining data privacy. We prove that the problem of finding the *globally optimal safe and faithful view*, i.e., the view that not only meets a set of privacy constraints but also is of the maximum data utility, is NP-hard. We propose the *locally optimal safe and faithful view* as the heuristic and show how we can efficiently find a locally optimal safe and faithful view in polynomial time. The details are discussed in Chapter 6.

The rest of thesis is organized as follows. In Chapter 2, we introduce the related background knowledge. Finally, we summarize the thesis in Chapter 7.
Chapter 2

Background

In this chapter, we introduce some background knowledge that is related to this thesis. We first briefly introduce XML and related terms, concepts, and techniques. Second, we introduce XML encryption. Finally, we introduce the basic concepts of $k$-anonymity, which is a popular technique for privacy-preserving data publishing.

2.1 eXtensible Markup Language (XML)

XML is an HTML-like language with an arbitrary number of user-defined tags. To be more specific, XML is a document with various pieces of text that are marked with tags, which are used to define the semantics of the text portions. The flexibility of definition of tags in XML makes it possible for different semantic layers to be built on top of data. Therefore XML can be tailored for different categories of applications. Since XML allows the Web users to use a common method for identifying and transforming heterogeneous data, it is used as the standard for information integration and exchange on the Web.

An XML element is a block of the form `<sometag>...</sometag>`, where `sometag` is the name of the self-defined tag. `<sometag>` and `</sometag>` are called the opening and closing tags, and the content between the opening tag and closing tag can be character data or sub-elements. Attributes can be assigned on XML elements. The main difference between elements and attributes is that elements are expressed in a structured form, which may be extensible, while attributes are always atomic tokens. For concreteness, consider the document in Figure 2.1, which is one possible XML representation of the students' course registration records. In the element `<Student StudID= "111">`, `Student` is the name of the element, `StudID` is the name of the attribute that belongs to the element `Student`. Its value is "111". We sometimes refer to element non-terminal.

Besides user-defined tags, XML remains semistructured. The document is structured in hierarchy by a sequence of elements. Elements directly (indirectly) nested within other elements are called children (descendants). For instance, element First in Figure 2.1 is a child of the element Name, and a descendant of element Student and Students. Conversely, element Name is the parent of element First, and elements Students and Student are ancestors of element First. The element Records is the root of the document.

An XML document is well formed (i.e., it can be interpreted by an XML
Figure 2.1: An XML Document of Student Record

parser) if the following conditions hold [60]: (1) It has a root element; (2) every opening tag is matched with a closing tag, and the elements are properly nested inside each other; (3) any attribute can occur at most once in a given opening tag, its value must be provided, and its value must be quoted. Any well-formed XML document can be represented as a tree. Specifically, every element in the XML document corresponds to a node in the tree. For any two elements $e_1$ and $e_2$, there exists an edge from the corresponding node of $e_1$ to that of $e_2$ if $e_1$ is the parent of $e_2$. Figure 2.2 is the tree representation of the XML document in Figure 2.1. We use "@" to specify attributes.

![XML Tree Example](image)

Figure 2.2: An Example of an XML Tree
Chapter 2. Background

2.1.1 Document Type Definition (DTD)

A Document Type Definition (DTD) is a context-free grammar that specifies the name of tags and attributes, as well as the structure of those tags/attributes in the XML document [60]. For instance, the DTD of the XML document in Figure 2.1 is shown in Figure 2.3. This DTD illustrates the most common DTD components: a set of ELEMENT and ATTLIST statements. For each tag (including the root tag in the document), there is a corresponding ELEMENT statement. Each ELEMENT statement specifies its tag and its children. Furthermore, for each tag that has attributes, the ATTLIST statement specifies the allowed attributes and their types. Besides ELEMENT and ATTLIST sets, DTD contains quantifiers to indicate the number of occurrences of the elements. Specifically, there are four kinds of quantifier, 1, ?, * and +, which indicate "exactly one occurrence", "zero or one occurrence", "zero or more occurrence" and "1 or more occurrence", respectively. For instance, for ELEMENT Student in Figure 2.3, there is a children list (Name, CrsTaken*). The * in this example indicates that there must be zero or more sub-elements CrsTaken of element Name.

Additionally, DTD contains a set of keywords. In particular, for the keywords shown in Figure 2.3, #PCDATA means that the element contains data that is going to be parsed by a parser. #REQUIRED(#IMPLIED, respectively) is for attribute declarations, which indicate that the attribute must always be provided (no default value is provided, respectively).

To be formal, DTDs are essentially a subset of (extended) context-free grammars, with productions of the form \( A \rightarrow \alpha \), where \( A \) is an element, and \( \alpha \) is a string over the vocabulary \( N \cup \{?, 1, +, *, (, (, \} \), \( N \) being the set of non-terminals. For instance, <!ELEMENT Student (Name, CrsTaken*)> can be transformed into \( Student \rightarrow Name, CrsTaken* \). The semantics of "|" in the production is to specify the optionality of the elements. For example, for <!ELEMENT Area (Asia | North America)>, the element Area may have either Asia or North America as its child, but not both at the same time. We
call productions containing "\|" disjunction productions. A DTD containing at
least one disjunction production is a disjunction DTD; otherwise, it is a disjunc
tionless DTD.

Throughout the thesis, we assume that the disjunction operator of a DTD is
not nested with disjunction or with quantifiers "\?, \*, +", and that quantifiers
are not nested with themselves more than once. There is no loss of generality
in this scenario, as any DTD can be rewritten into an equivalent DTD sat
ing this assumption, by introducing new non-terminals to factor out such
nested occurrences. Here, equivalence means that both grammars generate the
same language. In the production \( A \rightarrow \alpha_1 \mid \cdots \mid \alpha_k \), suppose \( \alpha_i \) is of the form
\(((\text{string}))\ell\), where \( \ell \in \{?, 1, *, +\} \); we then refer to \( \ell \) as the principal quanti
tifier associated with \( \alpha_i \). This notion also extends to single non-terminals and
we speak of the quantifier associated with a non-terminal in a sequence. The
following example illustrates this transformation and the ideas discussed above.

**Example 2.1.1 [Unnesting Disjunction]**

Consider the production \( A \rightarrow ((B?C^+)\mid (CD?)^+)\mid ((DEF)^+)\). This pro
duction is equivalent to the set of productions \( A \rightarrow G?, G \rightarrow H^* \mid I^+, H \rightarrow (B?C^+)\mid (CD?)^+, I \rightarrow (DEF)^* \). These productions do not involve a nesting of disjunction
with any quantifier or disjunction, and quantifiers are nested with quantifiers
at most once. The principal quantifier associated with \( (DEF)^* \) as well as with
\( (B?C^+)\) is \( '1' \). The quantifier associated with \( C \) in \( (B?C^+)\) is \( '1' \).

A DTD \( \Delta \) can be represented in the form of a directed node- and edge-
labeled graph \( \mathcal{G}(\Delta) = (V, E) \), called the DTD graph, as follows. The graph
\( \mathcal{G}(\Delta) \) contains an element node for each non-terminal, and the following la
celed edges: (i) for a disjunctionless production, \( A \rightarrow \alpha, \) edges \( (x, y) \) where \( x \)
is the non-terminal \( A \) and \( y \) is a non-terminal appearing in \( \alpha \); the label of
\( (x, y) \) is the quantifier associated with \( y \) in \( \alpha \); (ii) for a disjunction production
\( A \rightarrow [\alpha_1 \mid \cdots \mid \alpha_k] \), an edge \( (x, y) \) with label '1' where \( x \) is the non-terminal \( A \) and
\( y \) is the disjunction node associated with this production; (iii) for a disjunction
production \( A \rightarrow [\alpha_1 \mid \cdots \mid \alpha_k] \), an edge \( (x, y) \) with label '1' where \( x \) is the disjunc
tion node associated with this production and \( y \) is one of the non-terminals
\( \alpha_i, 1 \leq i \leq k \).

Figure 2.4 shows a sample DTD graph and its simplified version. Often,
it is useful to simplify a DTD graph without losing essential information. We
drop the label '1' and use it as the default label of an edge. When a production
does not involve disjunction, we can eliminate the disjunction node. Addition
cally, if the principal quantifier associated with the sequence in a disjunctionless
production is '1', we can eliminate that sequence node. In effect, we merge the
disjunction and sequence nodes with the non-terminal node of the production.
Figure 2.4(b) shows the simplified DTD graph. In our examples, we usually use
simplified DTD graphs.

We call a DTD a tree (respectively, acyclic, cyclic) provided its DTD graph
is a tree (respectively, is acyclic, is cyclic). Acyclic DTDs are also called DAG
(Directed Acyclic Graph) DTDs. An example of a DAG DTD graph is shown in
Figure 2.5. It is constructed based on part of the DTD of XMark [111], which
Chapter 2. Background

Figure 2.4: DTD Graph illustrated: (a) (Original) DTD graph and (b) simplified DTD graph of \(A \rightarrow (B?C^+)? \mid (CD^+), B \rightarrow (E F^+)\).

Figure 2.5: Example: DAG of XMark DTD

is a benchmark of XML database. In a DAG DTD graph, a node can have multiple parents. For example, the node Price in Figure 2.5 is a child of both Book and Software. Thus in the matching instance of this DTD, both Book and Software elements will have a child element Price. However, these two Price elements may have different data values.

2.1.2 XML Tree Pattern Queries (TPQ)

To selectively access the portion of the XML document, several query languages have been proposed by the W3C (The World Wide Web Consortium) organization. XPath [112] is a popular XML query language. XPath addresses parts of an XML document. It also provides basic facilities for manipulation of strings, numbers and booleans. The primary syntactic construct in XPath is the path expression, which will be evaluated. A node set, a string, a number, or a boolean will be returned as the result of evaluation. For instance, the query //Student[/Name/First= 'Wendy']//CrsTaken/@Semester will return the course registration information of students whose first name is Wendy. Here
Chapter 2. Background

14

Student
  Name
  First
  'Wendy'

Figure 2.6: Examples of Tree Pattern Query

"/" (or child axis) indicates a parent-child relation and "/" (or descendant axis) indicates an ancestor-descendant relation between elements. Besides "/" and "/" predicates, XPath also include other structural predicates. For example, the following-sibling axis returns all the following siblings of the context node, the preceding-sibling axis returns all the preceding siblings of the context node, and the following axis returns all nodes in the same document as the context node that follow the context node in the document order. In this thesis, we focus mainly on the "/" and "/" predicates.

Tree pattern queries abstract the core fragment of all XML query languages. A tree pattern query (TPQ) is a node-labeled and edge-labeled rooted tree with its edges labeled by pc (for parent-child relation) or ad (for ancestor-descendant relation), with nodes labeled by an element tag, and with leaves labeled by either an element tag or a constraint that constrains the element value. A subset of nodes is marked as distinguished, which abstract the notion of output nodes of the query and are shown inside boxes in our examples. The pc-edges and ad-edges are indicated by single and double lines respectively. Figure 2.6 shows the tree representation of the query //Student[/Name/First='Wendy']//CrsTaken/@Semester.

Answers to a TPQ are based on matchings, which are functions that map pattern nodes to data nodes, while satisfying node predicates (such as tag) and relationships between nodes. For example, by applying the tree pattern query in Figure 2.6 to the database in Figure 2.1, the value "S2007", which is the value of attribute semester of the student whose first name is Wendy, is returned as the answer.

2.1.3 Structural Join: Efficient Evaluation of XML Queries

As shown in Section 2.1.2, XML queries typically specify patterns of selection predicates on multiple elements that have some specified tree-structured relationships. The complex patterns can be decomposed into a set of parent-child and ancestor-descendant relationships between pairs of query nodes. Finding all occurrences of these basic structural relationships in XML database is clearly a core operation in XML query processing. The structural join is designed for this task [9]. It focuses on the containment (ancestor-descendant or parent-child) relationship of the XML elements to be joined. The basic idea is quite simple:
Chapter 2. Background

Figure 2.7: Examples of 3-tuple Index for Structural Join

Every element in the XML database will be assigned a 3-tuple index \((\text{DocID}, \text{StartPos}, \text{EndPos}, \text{LevelNum})\) to represent its position in the database [9]. DocID is the identifier of the document. StartPos and EndPos are generated by doing a depth-first traversal of the document tree whose identifier is DocID, and sequentially assigning a number during the traversal. LevelNum is the nesting depth of the element (or string value) in the document. Figure 2.7 shows an example of XML tree that is labeled with 3-tuple index. Based on the index, the structural relationships between tree nodes (elements or string values) whose positions are recorded in this fashion can be efficiently determined:

1. **Ancestor-Descendant**: A tree node \(n_2\) whose index is \((D_2, S_2, E_2, L_2)\) is a descendant of a tree node \(n_1\) whose index is \((D_1, S_1, E_1, L_1)\) if and only if \(D_1 = D_2\), \(S_1 < S_2\), and \(E_2 < E_1\).

2. **Parent-Child**: A tree node \(n_2\) whose index is \((D_2, S_2, E_2, L_2)\) is a descendant of a tree node \(n_1\) whose index is \((D_1, S_1, E_1, L_1)\) if and only if \(D_1 = D_2\), \(S_1 < S_2\), and \(E_2 < E_1\), and \(L_1 + 1 = L_2\).

For example, the ancestor-descendant relationship between node A and node C in Figure 2.7 can be easily determined by the above ancestor-descendant reasoning. There are numerous works on efficient implementation of structural joins (for example, [61], [9], [24]) for XML query evaluation. More details regarding structural joins and other techniques of XML query evaluation can be found in [48].

2.1.4 XPath Query Containment

XPath query containment has many applications in XML querying, integration and transformation. One example of applications is query minimization, which we will mention in Chapter 3. The problem definition of query containment is following: given two XPath queries \(Q_1\) and \(Q_2\), we say \(Q_1\) contains \(Q_2\) if for every XML document \(D\), the output of \(Q_1\) on \(D\) contains the output of \(Q_2\) on \(D\). There has been much work done on this problem, for example, [5], [76], and [93]. These papers examined the complexity of checking containment mapping for different XPath fragments and under various setups. A summary of the complexity results and algorithmic techniques of XPath query containment can be found in [81].
2.2 XML Encryption

Encryption is one of the techniques used to protect the confidentiality of electronically stored information. The W3C has recommended a standard of XML encryption [116]. It defines the XML vocabulary and processing rules that enable confidentiality to be applied to a variety of content, which serves the purpose of maintaining the confidentiality of XML data both while in transit and when stored. There are three kinds of granularity for XML encryption: encrypting an XML element, encrypting XML element content that is also elements, and encrypting XML element content that is character data. By encrypting an XML element, the content as well as the identity of the element itself must be hidden. If the root element is encrypted, the whole document will be encrypted as well. By encrypting XML element contents that are also elements, all descendant elements will be encrypted recursively. By encrypting XML element content that is character data, only the character data will be encrypted. When an XML element or element content is encrypted, it is replaced by an &lt;EncryptedData&gt; element. XML encryption supports the selection of appropriate encryption algorithms. It also supports super-encryption, which is encryption on encrypted data.

Example 2.2.1 [XML Encryption]
Consider the payment information shown in Figure 2.8 (a). Since Smith's credit card number is sensitive information, it can be kept confidential by encrypting the CreditCard element. The encryption result is shown in Figure 2.8 (b). Another choice is to encrypt the content of the Number element. The encryption result is shown in Figure 2.8 (c).

There are several commercial products available that provide element-wise encryption to transactions involving XML data. IBM's AlphaWork's XML Security Suite[101] is a tool that provides security features such as digital signature, encryption, and access control for XML documents. VisualSafe Corporation has implemented a product named “VisualSoft XMLSecure”[115], which is a powerful Component Object Model (COM) component that provides cryptographic features such as digital signatures, verification, encryption and decryption in any XML framework.

2.3 K-anonymity

There has been tremendous growth in the amount of personal data that can be collected and analyzed from the Web in recent years. Data mining tools are increasingly being used to infer trends and patterns of the collected data. In many scenarios, access to large amounts of personal data is essential in order for accurate inferences to be drawn. However, the use of data containing personal information has to be restricted in order to protect individual privacy. Thus the data has to be released in a way that enables one to analyze the data patterns and trends without violating the privacy of individual records.
Chapter 2. Background

(a) The Original XML Database

```xml
<?xml version='1.0'?>
<PaymentInfo xmlns='http://example.org/paymentv2'>
  <Name>John Smith</Name>
  <CreditCard Limit='5,000' Currency='USD'>
    <Number>4019 2445 0277 5567</Number>
    <Issuer>Example Bank</Issuer>
    <Expiration>04/02</Expiration>
  </CreditCard>
</PaymentInfo>
```

(b) Encrypt Element CreditCard

```xml
<?xml version='1.0'?>
<PaymentInfo xmlns='http://example.org/paymentv2'>
  <Name>John Smith</Name>
  <CreditCard Limit='5,000' Currency='USD'>
    <Number>
      <EncryptedData Type='http://www.w3.org/2001/04/xmlenc#Element'
                      xmlns='http://www.w3.org/2001/04/xmlenc#'>
        <CipherData>
          <CipherValue>A23B45C56</CipherValue>
        </CipherData>
      </EncryptedData>
    </Number>
    <Issuer>Example Bank</Issuer>
    <Expiration>04/02</Expiration>
  </CreditCard>
</PaymentInfo>
```

(c) Encrypt Content of Element Number

```xml
<?xml version='1.0'?>
<PaymentInfo xmlns='http://example.org/paymentv2'>
  <Name>John Smith</Name>
  <CreditCard Limit='5,000' Currency='USD'>
    <Number>
      <EncryptedData xmlns='http://www.w3.org/2001/04/xmlenc#'
                      Type='http://www.w3.org/2001/04/xmlenc#Content'>
        <CipherData>
          <CipherValue>A23B45C56</CipherValue>
        </CipherData>
      </EncryptedData>
    </Number>
    <Issuer>Example Bank</Issuer>
    <Expiration>04/02</Expiration>
  </CreditCard>
</PaymentInfo>
```

Figure 2.8: Various Encryption Schemes to Protect Credit Card Number

One approach to privacy-preserving data publishing is suppression of some of the data values, while releasing the remaining data values exactly. However, suppressing just the identifying attributes is not sufficient to protect privacy. For example, consider the table in Figure 2.9(a), in which the identifying attribute name is removed. By joining this table with public databases (such as a voter list), non-identifying attributes in the above table such as Age, Race, Gender and ZipCode, can be used together to identify individuals. In fact, Sweeney [83] observed that for 87% of the population in the United States, the combination of Date of Birth, Gender and Zip Code corresponded to a unique person. This is called record linkage.

In order to ensure the protection of privacy against record linkage, k-anonymity
Chapter 2. Background

<table>
<thead>
<tr>
<th>Age</th>
<th>Race</th>
<th>Gender</th>
<th>ZipCode</th>
<th>Disease</th>
</tr>
</thead>
<tbody>
<tr>
<td>40</td>
<td>White</td>
<td>Male</td>
<td>21004</td>
<td>Diarrhea</td>
</tr>
<tr>
<td>35</td>
<td>White</td>
<td>Female</td>
<td>21004</td>
<td>Flu</td>
</tr>
<tr>
<td>27</td>
<td>Hispanic</td>
<td>Female</td>
<td>92009</td>
<td>Flu</td>
</tr>
<tr>
<td>27</td>
<td>White</td>
<td>Female</td>
<td>92009</td>
<td>Diarrhea</td>
</tr>
</tbody>
</table>

(a): Base Table T

<table>
<thead>
<tr>
<th>Age</th>
<th>Race</th>
<th>Gender</th>
<th>Zip Code</th>
<th>Disease</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>White*</td>
<td></td>
<td>21004</td>
<td>Diarrhea</td>
</tr>
<tr>
<td>*</td>
<td>White*</td>
<td></td>
<td>21004</td>
<td>Flu</td>
</tr>
<tr>
<td>27</td>
<td>*</td>
<td>Female</td>
<td>92009</td>
<td>Flu</td>
</tr>
<tr>
<td>27</td>
<td>*</td>
<td>Female</td>
<td>92009</td>
<td>Diarrhea</td>
</tr>
</tbody>
</table>

(b): A 2-anonymity table

Figure 2.9: Example of K-anonymity

was proposed and has become a popular approach in privacy-preserving data publishing. The goal of k-anonymity is to guarantee that every individual object is hidden in a crowd of size k. A release of data is said to adhere to k-anonymity if each released record has at least (k-1) other records also visible in the release whose values are indistinct over a special set of fields, which are called quasi-identifiers[84]. The quasi-identifier contains those fields that are likely to appear in other known data sets. For example, for the base table shown in Figure 2.9 (a), the quasi-identifier attributes are \{Age, Race, Gender, ZipCode\}. To achieve k-anonymity, some data values will be suppressed and some will be generalized. Generalization entails replacing an entry value with a less specific but semantically consistent value, for example, generalize city to region. By suppression/generalization, for each tuple in the modified table, there are at least \(k\geq 1\) other tuples in the modified table that are identical to it along the quasi-identifying attributes. For example, Figure 2.9 (b) is a 2-anonymity table such that for each group of tuples with the same values on \{Age, Race, Gender, ZipCode\}, it contains at least two tuples. Thus, even if an attacker has access to all the quasi-identifying attributes of all the individuals represented in the table, he would not be able to track down an individual’s record further than a set of at least 2 records, in the worst case. The privacy parameter \(k\) is chosen according to the application in order to ensure the required level of privacy.

There has been extensive research done on k-anonymity. It has been shown that the problem of k-anonymization by suppressing cells in the table is NP-hard [69]. Approximation algorithms have been proposed in [4]. Much work has been done on creating efficient algorithms for k-anonymity using generalization and tuple-suppression techniques (for example, [12, 58]). A different formal definition of privacy was proposed in [26] for published data based on the notion of blending in a crowd. Based on the definition of k-anonymity, new notions such as l-diversity [65] and t-closeness [62] have been proposed to provide improved privacy. Specifically, the notion of l-diversity guards against limited amounts
of background knowledge unknown to the data publisher. The main idea is to require that, for each group, the \( l \) most frequent sensitive values are roughly equi-probable, so that it ensures that the attacker needs at least \( l - 1 \) pieces of background knowledge to sufficiently disclose the sensitive value of any individual. The notion of \( t \)-closeness overcomes some shortcomings of \( l \)-diversity (for example, \( l \)-diversity is insufficient to prevent attack when the overall distribution is skewed). It requires that the distribution of a sensitive attribute in any equivalence class is close to the distribution of the attribute in the overall table. The distance between the two distributions should be no more than a threshold \( t \).

Anatomy [87] and bucketization [68] are two anonymization techniques that have recently been proposed. Both of them do not require data generalization. Anatomy releases all the quasi-identifier and sensitive values directly in two separate tables. Combined with a grouping mechanism, this approach protects privacy, and captures a large amount of correlation in the original database. Bucketization is to partition the tuples in the original database into buckets, and then to separate the sensitive attribute from the non-sensitive ones by randomly permuting the sensitive attribute values within each bucket. Anatomy corresponds exactly to the notion of bucketization.
Chapter 3

Secure and Efficient Query Evaluation over Access-controlled XML Databases

The rapid emergence of XML as a standard for data representation and exchange over the Web has sparked considerable interest in models and efficient mechanisms for controlled access, to information represented as XML (for example, [15, 16, 18, 19, 28, 34, 35, 95]). When accessing XML data using queries, a major source of inefficiency in the evaluation of queries is the considerable overhead spent on checking the security level associated with data elements that are accessed by the query, to see if the user is authorized to access them [28]. Thus, it is important to optimize the evaluation of queries over secure databases so as to minimize the number of such security checks.

Cho et al. [28] addressed this problem for tree pattern queries (essentially the subclass of XPath queries that use the child and descendant axis), for the XML multi-level security (MLS) model [52], focusing on tree-structured schemas. Since the XML MLS model supports inheritance of security levels (down the XML tree) with overriding, checking the security level associated with an element would, in general, require (recursively) traversing through the chain of ancestors of an element to find the nearest ancestor with an explicitly defined security level. Cho et al. [28] showed that such recursive checks are often redundant, and proposed polynomial-time algorithms that took advantage of the structure of the scheme to either eliminate security checks at specific query nodes, or replace the recursive checks with (cheaper) local checks at query nodes.

While the contribution of [28] in dealing with tree schemas is important, most realistic schemas are not tree structured. There is considerable sharing of element types in schemas. Consider, for example, the schema in Figure 3.1, based on the XMark DTD. Both Book and Account elements have ID elements as children. Because of element sharing, real-life schemas tend to be acyclic (or DAG) schemas, sometimes with disjunction nodes\(^1\). Since there can be exponentially many schema paths between a pair of nodes, determining how to optimize the recursive checks on query nodes needs to reason about sets of schema paths, in-

\(^1\)For example, the U.S. Congress bills dataset we tested had many disjunction nodes.
Figure 3.1: Example XMark-based DAG DTD

stead of just a single schema path as in the case of tree-structured schemas. For example, a simple XPath query: `/OnlineSeller[.//ID and .//Price]` has to reason about multiple paths between `OnlineSeller` and `ID` nodes in the schema, and their relationship to the multiple schema paths between `OnlineSeller` and `Price` nodes. How can one do this reasoning efficiently over DAG DTDs? A straightforward extension of the algorithm of [28], reasoning over sets of schema paths, would lead to an exponential-time algorithm for the case of DAG schemas. In this thesis, we study the problem of efficiently (in polynomial-time) optimizing security checks over tree-pattern queries against DAG (i.e., acyclic) schemas, and make the following contributions.

• We propose a sound and complete set of rules for optimizing security checks on tree-pattern query nodes against DAG DTDs without disjunction nodes, and develop efficient, polynomial-time algorithms for this task (Section 3.4).

• There is an additional source of complexity regarding this problem when the schema contains disjunction nodes, since a disjunction DTD is logically equivalent to the disjunction of a potentially exponential number of "candidate" DAG DTDs without disjunction. The optimization we perform needs to be correct with regard to all database instances that are valid with regard to any one of these candidate DTDs. We identify how the rules for the DTD without disjunction nodes can be enhanced to deal with the DTD with disjunction nodes, and then develop efficient, polynomial-time algorithms for the task of security-check optimization, based on a novel strategy for dealing with the combinatorial explosion (Section 3.5).

• Finally, we report on an extensive set of experiments we ran to measure the utility and effectiveness of security-check optimization on various benchmark data sets and DTDs. Our experiments complement our analytical results, and demonstrate that security-check optimization can make an order of magnitude difference in query evaluation performance, while incurring negligible overhead for optimization (Section 3.6).
Next, we introduce the related work of this chapter in Section 3.1 and formally state the problem we address in this chapter in Section 3.2. The definitions and conventions we use in this chapter are given in Section 3.3. We summarize this chapter in Section 3.7. This work was originally published as a chapter in the book "Security in Decentralized Data Management" [88].

3.1 Related Work

Recent research on XML security has focused considerable attention on the complementary problems of: (i) development of sophisticated access-control models, and (ii) efficient mechanisms to enforce policies in these models. We describe recent work on these two problems in this section.

3.1.1 XML Access-Control Models

A number of security models have been proposed for XML (see [43] for a recent survey). Bertino et al. [18-20] defined an access-control model that is capable of describing a set of access-control policies ranging from multiple documents to specific elements within a document. The access control policies are based on the notion of subject credentials and on policy specifications at different granularity levels, with different propagation options. A credential is a set of attributes concerning a subject that is relevant for security purposes. Credentials of similar structure are grouped into the same credential type. The credentials and credential types are encoded by an XML-based language (more details concerning this language can be found in Bertino et al. [17]). Access-control policies are specified both at the DTD and the database level. Policies can be specified on the granularity of whole DTDs (respectively, databases) or on selected elements and/or attributes within DTDs (respectively, databases). Elements/attributes can be either explicitly specified, or can be selected on the basis of their content. The access-control language used to specify authorization policies is a variant of XPath and is based on the composition of element names and attribute names. The semantics of access control to a user is a particular view of the documents determined by the relevant access-control rules. An idea similar to this access-control model is used in the implementation of the Author-X prototype [15, 16].

Damiani et al. [35, 37] proposed an access-control model for XML documents and schema, which allows positive and negative authorizations and authorization propagation. Authorizations specified on an element can be defined as applicable to the element's attributes only, or, in a recursive approach, to its sub-elements and their attributes. Damiani et al. developed an approach for expressing access-control policies using XML syntax. The semantics of this access-control model has the same flavor of the view idea proposed in [16]. In the same framework, Kudo et al. [56] stressed richer authorization decisions by incorporating the notion of provisional authorization into traditional authorization semantics, where the user's requests for data will be authorized provided he/she takes certain security actions such as signing his/her statement prior
to authorization of his/her request. It enables the client not only to browse XML documents securely but also to update each document element securely. This provisional model provides an XML access-control language called XACL that integrates several security features such as authorization and audit trail in unified manner.

Anutariya et al. [10] explored a rule-based XML access-control model. They used an XML-based rule language, namely XML Declarative Description (XDD), for authorization and policy definitions. XDD extends ordinary XML elements by incorporation of variables so that the expressiveness and representation of implicit information are enhanced. By means of XDD, the model permits one to describe declaratively fine-grained access control and to define multiple, distinct policies to be enforced on different authorization definitions. In addition, they define a special type of query, named access evaluation query, which returns an access decision (grant or deny) by retrieving the matching specified authorization rules from the authorization database and applying an appropriate policy to resolve conflicts or to yield a default permission.

Wang et al. [91] presented a role-based approach to access control that uses the role graph model and the object-oriented database authorization model as its basis. In this model, the users are grouped by roles. In particular, based on the role graph model, the access-control model ensures that when a privilege is added to a role, all implied privileges are added as well. The privileges on the roles can be associated with any part of an XML document, a single attribute or element value, or a set of documents. In combination with the ideas of the object-oriented database authorization model, the XML schema could be used to construct the authorization object schema, and the XML documents could be used to construct the authorization object graph. With the aid of the role graph model and the object-oriented database authorization model, a complex authorization model can be built and enforced.

Goel et al. [45] presented an XQuery-based approach for specifying fine-grained access control in XML based on the relationship of a document to other documents. Access control can be specified in terms of access on static portion of the database. The content relationships between documents can be used to specify policy at the schema level. However, there is no implementation of this security model.

Recently, Cho et al. [28] proposed a simple multi-level security (MLS) model for XML data, where individual XML elements may each have an explicitly associated security level (represented as an attribute of that element), as do all users. If an element does not have a security level explicitly associated with it, it inherits the security level associated with the closest ancestor where it is explicitly defined. To identify which elements must, may or cannot have security levels explicitly associated with them, the model of [28] allows the DTD (schema) to be labeled with this information.

Our contributions are complementary to this body of work, since our techniques are applicable to a wide variety of access control models (including the ones mentioned above) that support inheritance of security levels (down the XML tree) with overriding.
3.1.2 Efficient Enforcement Mechanisms

Most of the existing mechanisms for authorization policy enforcement can be categorized as: (1) authorization views, (2) access-control index, (3) static analysis of the queries, and (4) query rewriting.

3.1.2.1 Authorization Views

Fan et al. [42] proposed an approach to access-control enforcement based on security views. A security view is a portion of the original document and the underlying document DTD that exposes exactly the necessary schema structure and document content to authorized users. Authorized users can only pose queries over the security view by making use of the exposed DTD view to formulate their queries. The authors proposed a novel quadratic time algorithm that, given the document DTD and an access-policy specification, automatically derives a sound and complete security-view definition (i.e., a view that exposes all and only accessible data elements and schema structure).

To elaborate on certain issues left open in [42], Kuper et al. [57] studied access control and security specifications defined on general DTDs in terms of regular expressions, rather than the normalized DTDs employed in [42]. They developed a new algorithm for deriving a security view definition from a more intuitive access-control specification (with regard to a non-recursive DTD) without introducing dummy element types (the fake element without content) as in [42], thus preventing inference of sensitive information from the XML structure revealed by dummies. They also developed an algorithm for materializing security views, which ensures that views conform to the secure DTD views.

It is quite expensive to actually materialize and maintain multiple security views of a large XML document. To avoid constructing multiple views, Bertino et al. [20] used encryption to enforce the authorization policies on one XML document. They used different keys for encrypting different portions of the same document. Each portion is encrypted with one and only one key. The same (encrypted) copy of the document is then broadcast to all subjects, whereas each subject only receives the key(s) for the portion(s) he/she is enabled to access. The encryption granularity of the document is of the finest level, i.e., every single attribute or element of the document is encrypted with a potentially different key. To limit the number of keys that will be generated, a method was proposed by which the portions of the document to which the same access policies apply are encrypted with the same keys.

Miklau et al. [71] also proposed a framework for enforcing access-control policies on published XML documents using cryptography. In this framework, each node in the XML document is guarded by positive boolean formulas, namely a protection, over a set of cryptographic keys, instead of only one key, as in Bertino et al. [20]. The access control policies are translated into protections and are applied to the XML document by encryption. The authors describe a simple extension of XQuery that allows the user to access data selectively, and moreover, to guide the query processor to locate which keys to use where. Based
on their work, Abadi et al. [1] provide a precise justification of the security guarantee of this framework against probabilistic polynomial-time adversaries with respect to a strong, computational notion of security.

Damiani et al. [35, 37] used a different approach for the implementation of the authorization view. The view that is accessible to a user is computed through a tree labeling process. The tree labeling process associates a tuple to each node as a label, whose content reflects the authorizations specified on the node. The labeling of the complete document can be obtained by starting from the root and proceeding downwards with a preorder traversal.

3.1.2.2 Access-control Index

Another method of enforcing access-control policies on an XML document is to represent the access-control information by some data structure, i.e., the index scheme, so that the accessibility of (a portion of) the XML data can be efficiently determined by exploiting the index.

Bouganim et al. [22] studied how to implement an access control model on streaming XML documents. The authors proposed an efficient access-control rule evaluator in which each access-control rule (in XPath syntax) is represented by a non-deterministic finite automaton. This evaluator benefits from the design of a highly compact index structure, called a skip index, which allows fast access to the authorized portion of the document and skips the irrelevant parts with respect to the access-control policy and a potential query. The skip index is encoded recursively into the XML document to allow streaming, which can be used as a new means of compressing the structural part of the document.

Gummadi et al. [49] proposed a three-dimensional bitmap data structure, named the security cube, to represent the access-control policies. The three dimensions of the security cube are documents, element paths and user/group. It is fine-tuned to represent the access control at the element path level of granularity (the lowest level of granularity) for multiple users. Thus, the security cube can be effectively used to provide control of access to multiple users over several documents with many element paths. The security cube supports quick access, and is flexible for maintenance and updates. It is extendible to include new subjects, for example, Internet Protocol (IP) address or symbolic names, to add write and append privileges, and to express the access control on the content level. The authors also showed how to enhance the security cube in different environments, for example, in the mandatory access-control model, discretionary access-control model, role-based model and rule-based model.

Yu et al. [95] investigated the problem of efficiently determining whether or not a user has access to a particular XML element for arbitrary access-control policies. A compressed accessibility map (CAM) was proposed to resolve the problem in a way that compactly represents the data items by exploiting the locality of accessibility in hierarchical XML data. The authors presented a CAM lookup algorithm for determining if a user has access to a data item. The complexity of the lookup algorithm is proportional to the product of the depth of the item in the XML data and logarithm of the CAM size. Yu et al. also developed
an algorithm for building an optimal size CAM that takes time linear in the size of the XML data set. While optimality cannot be preserved incrementally under data item updates, they provided an algorithm for incrementally maintaining near-optimality.

Zhang et al. [96] presented a compact XML access-control labeling scheme that exploits both access-control structural locality within the XML data and correlations between access rights of different users. The physical representation of the access-control labeling is a compact index scheme that can be constructed by using a single pass over an XML document with access-control labels. This physical scheme can be integrated with a next-of-kin (NoK) XML query processor so that efficient and secure query evaluation is supported. The authors noted that all the updates to the access-control labeling are local, i.e., the updates are confined within a contiguous region of the affected data.

3.1.2.3 Static Analysis
To securely apply a query on the access-controlled data, the accessibility of each query node must be checked against the database. However, such a process can be expensive for XML documents of large size. Murata et al. [74] studied the problem of how to determine whether a query expression is safe with regard to the given access-control policy by static analysis, i.e., without evaluating any query expression against an actual database. Run-time checking is needed only when static analysis is unable to determine whether the access is granted or denied. The key idea behind this approach is to construct the automata for queries, access-control policies and schemas individually. By comparing these three automata, the queries that do not satisfy access-control policies are filtered out, and the “safe” queries are referred to the XML query engine for further checking and processing. If there is no schema available, the schema automata will be skipped and the comparison will be made only between access control automata and query automata.

3.1.2.4 Query Evaluation and Rewriting
It has been noted that materializing and maintaining multiple security views of a large XML document is expensive. To avoid the overhead of view materialization and the complexities of view maintenance, Fan et al. [42] proposed a novel XML query-rewriting algorithm that transforms an input XPath query posed over a security view to an equivalent, secure query over the original document; this procedure yields an effective querying mechanism that completely bypasses view materialization. The authors also introduced a new query optimization technique that leverages the structural properties of DTDs to optimize the evaluation of the transformed query.

Luo et al. [64] examined three different approaches, namely primitive, pre-processing and post-processing processes, to perform the access control policy on data of a finer granularity. The primitive approach is to view both query and access control policies as two constraints and apply both constraints to the
database. The pre-processing approach is to first prune away the queries that conflict with the access control policies. Then the queries are rewritten with regard to the access control policies so that only the safe "revised" queries will be applied on the database. This rewriting is based on a method called QFilter, which in turn is based on non-deterministic finite automata. The post-processing approach is to first apply all queries to the database, no matter whether they are safe or not. Then the returned results will be verified by the post-filtering process and the unsafe answers will be pruned away. Through their experimental results, the authors showed that the QFilter-based pre-processing approach significantly outperforms the primitive and post-processing approaches.

Our focus is on finding optimal, correct rewritings of queries against an XML database conforming to a given DAG DTD. The rewriting guarantees secure evaluation of the queries. Our model supports inheritance and overriding of security levels. The rewriting is accomplished by means of adornning query tree nodes with security check annotation labels that infer the accessibility of a given node. Our work is inspired by, and substantially enhances, the work done by Cho et al. [28], which focused on much simpler tree-structured DTDs. A straightforward extension from tree-structured schema to DAG schema by reasoning over sets of schema paths would lead to an exponential-time algorithm. We study the problem of efficiently (in polynomial-time) optimizing security checks over tree pattern queries against DAG (i.e., acyclic) schemas without disjunction nodes. There is an additional source of complexity when the schema contains disjunction nodes, since a disjunction DTD is logically equivalent to the disjunction of a potentially exponential number of "candidate" DAG DTDs without disjunction. We develop efficient, polynomial-time algorithms for the task of security check optimization based on an elegant strategy for dealing with the combinatorial explosion (Section 3.5).

### 3.2 Model and Problem

#### 3.2.1 Security and Authorization Model

We adopt a very general security model, where any mechanism (e.g., multi-level security, role-based access, or rules) can be used to explicitly determine which data object (XML element) is accessible by which user/subject. Since it can be too tedious and cumbersome to require every element to have its security (i.e., accessibility) explicitly defined, the various proposals for XML security permit an element (whose security is not explicitly defined) to inherit its security from the element's closest ancestor where security is defined. We require that the root element must have its security explicitly defined. This security model permits the security of elements to be defined in a flexible way, allowing them to be inherited and overridden in an arbitrary fashion. Attribute-level security can be handled analogously to that for elements, and is not considered further.

---

2 Our primary focus being query processing, we consider only read access.
[28] proposed a lightweight extension to DTDs whereby the DTD nodes can be associated with security information, declared as:

- **Mandatory** (M) (all instances must have their security defined explicitly),
- **Optional** (O) (instances may have their security defined explicitly or inherited),
- **Forbidden** (F) (no instance may have its security defined explicitly; only inheritance is permitted).

We follow the same security model. We require that the root of the schema must be specified as (M) (i.e., the security level of the root must be defined explicitly). Let us look at an example of a DTD with security annotation (as shown in Figure 3.2). Since the accounts information is sensitive, it must have its security level explicitly specified. Therefore, the annotation of the Accounts node is declared as (M). Furthermore, the security level on each account varies for different customers. It is either the same security level as specified on the ancestor elements, or is specified locally. Therefore, the annotation of node Account is declared (O), as is the Cart node. Finally, the security levels of all other schema nodes are labeled with (F) because their corresponding instance nodes do not need to have security levels explicitly defined. The flexibility of the security annotations allows our model to be applicable to a role-based access control environment as well.

### 3.2.2 Multi-level Security and Authorization

We assume a set of security levels, which may be any finite partially ordered set. For simplicity of exposition, we use total ordered security levels in all our examples. For instance, in a military environment, it is common to use the security levels \( S = \text{unclassified, confidential, secret, top secret} \), with the ordering...
unclassified < confidential < secret < top secret. The security levels used in a commercial application may vary, and depend on the organization’s policy.

The basic idea behind multi-level security is that each resource/object (e.g., XML element) is potentially assigned a security level, as is each user/subject that is authorized to access objects in the space (e.g., a collection of XML documents). Here, a “user” might be an application or a human. A subject may only access an object if the security level assigned to that subject is no less than that assigned to the object. Thus, a subject whose security level is “secret” can access (i.e., read) objects with security levels “secret”, “confidential”, or “unclassified”.

In the context of XML data trees, security levels are assigned to elements by associating a separate attribute called, for example, SecurityLevel. Thus, a DTD needs to permit such attributes in addition to the usual content of elements. Since XML is tree structured, the security level of an element, if not specified, can be inherited from its closest ancestor. On the other hand, a security level defined at an element always overrides the inherited one. This overriding can be monotone, in that security levels defined at sub-elements are never less than those defined at their parent, or non-monotone, where no such restriction is imposed. Clearly, non-monotone overriding is more expressive, in that it permits a broader class of applications. For this reason and for brevity, in the following, we only consider non-monotone overriding.

### 3.2.3 SC Annotations and Problem Statement

Secure evaluation of a tree pattern query involves checking the (explicitly defined or inherited) security of every data tree node matching a query node to ensure that it is accessible by the user. [28] has introduced the predicates at every query node by examining the value of the SecurityLevel attribute of the matching tree node in the data. These notions are RC (recursive check), LC (local check), and NC (no check); we term them security check annotation labels (SC annotation labels). LC merely amounts to checking the value of the SecurityLevel attribute at a given node, while NC is a no-op. RC, on the other hand, involves recursively checking the value of this attribute at every node starting from the given node until a nearest ancestor is reached where the attribute is defined. Thus, every tree pattern query with SC annotation labels added can be expressed as a query with function calls added for local and recursive checks and can be evaluated using the same evaluation engine. To formulate the optimization problem, we provide the following definition:

**Definition 3.2.1 [Minimal SC Annotation]** Let Q be a tree-pattern query. A security check (SC) annotation of Q is essentially Q with each of its nodes associated with one of the SC annotation labels RC, LC, NC. Define RC > LC > NC, reflecting the complexity of performing these checks in general. Given two different SC annotations $A_1, A_2$ of a tree-pattern query Q, we say $A_1$ dominates $A_2$, written $A_1 > A_2$ provided that for every node $x \in Q$, $A_1(x) > A_2(x)$. 
Chapter 3. Query Evaluation over Access-controlled XML Databases

An SC annotation $A$ in a set of annotations $S$ is minimal provided there is no annotation $B \in S$ such that $A > B$.

We now formalize our notion of correctness. Let $Q$ be a query issued by a user against a database $D$. A tuple of data nodes that are mappings of pattern nodes of $Q$ is a binding tuple. We consider a binding tuple $(d_1, ..., d_k)$ associated with $Q$ and $D$ to be safe provided each $d_i (1 \leq i \leq k)$ is accessible to the user. An answer tuple is safe provided it is the projection of a safe binding tuple.

**Definition 3.2.2 [Correct Annotation]** Given a query $Q$ and a DTD $\Delta$, an annotation $A$ of $Q$ is correct with regard to $\Delta$, provided that on every valid database instance $D$ of $\Delta$, every answer to $Q$ on $D$ obtained using $A$ is safe.  

There is a trivial way to make sure we get exactly the safe and correct answers: annotate all query nodes with RC, so that the security level of every node will be checked no matter whether it inherits the security levels or not. This procedure guarantees that the evaluation of $Q$ never accesses nodes the user with security level $\text{usl}$ is not allowed to access. However, it is very expensive to evaluate queries this way, since many avoidable redundant security checks may be performed. Thus, we seek to optimize the SC annotations on as many query nodes as possible. More precisely, we give the problem statement as follows.

**Problem Statement:** Given a tree-pattern query $Q$ and a DTD $\Delta$, find a minimal SC annotation among the set of all SC annotations of $Q$ that is correct with regard to $\Delta$.

In this chapter, we assume that the queries $Q$ we consider are minimal, i.e., there is no query $Q'$ such that for any database $D$, $Q(D)$ contains $Q'(D)$ (see Section 2.1.4 for the definition of query containment). Minimizing tree-pattern queries is a well-researched topic and we refer the reader to [70, 93] for minimization techniques.

### 3.3 Definitions and Conventions

Before discuss the details of our approach, we provide the definitions and conventions that we will frequently use in this chapter.

Let $Q$ be a tree-pattern query, $\Delta$ be a DTD, and $x$ ($x'$, respectively) be a node in $Q$ ($\Delta$, respectively) with tag $t$. For a set of query nodes $S$, $S'$ denotes the corresponding set of DTD nodes. For simplicity, we blur the distinction between a (query or DTD) node and its tag, and consider node $A$ to mean a node with tag $A$. A node in $\Delta$ is overriding if it is marked (M) or (O) in $\Delta$. A DTD path from node $x'$ to $y'$ is overriding if it contains an intermediate node $z'$ ($z' \neq x', y'$) that is overriding, otherwise it is non-overriding. We say an

---

3 Note that the evaluation of an annotated query always produces all safe answers against any database. However, it may produce unsafe answers, when it is incorrect.

4 In specific examples, of course both the query node $x$ and the DTD node $x'$ will have the same tag, say $A$. 

overriding node \( z' \) exists between two DTD nodes \( x' \) and \( y' \) if it occurs on a path from \( x' \) to \( y' \), where \( z' \neq x', y' \). Such a node is closest to \( y' \) if there is a non-overriding DTD path from \( z' \) to \( y' \) (from \( x' \) to \( z' \)). We say a DTD node \( x' \) is an ancestor of a DTD node \( y' \) provided there is a DTD path from \( x' \) to \( y' \). We say a node \( w' \) is a lowest common ancestor of node \( x' \) and \( y' \) if (1) \( w' \) is the ancestor of \( x' \) and \( y' \), and (2) there does not exist a node \( z' \) that is the descendant of \( w' \) and the ancestor of both \( x' \) and \( y' \). Note that for any two nodes \( x \) and \( y \) in a DAG-structured DTD, there may exist multiple lowest common ancestors. However, for any two nodes in a tree-pattern query, their lowest common ancestor is unique.

A path in \( \Delta \) is optional if at least one of the edges in it is labeled either ‘?’ or ‘*’, otherwise it is non-optional. A path is unique if all edges in it are labeled ‘?’ or ‘1’. For a query node \( x \), its query parent is the node \( y \) in \( Q \) such that \( Q \) contains an (ancestor-descendant or parent-child) edge \((y, x)\). A query node \( x \) is unconstrained (i.e., has no constraint on it) if: (i) it is not a distinguished node (i.e., the output node) of \( Q \), (ii) \( Q \) has no value-based constraint of the form \( \text{tag} \ Op \ \text{value} \) (e.g., price>50) on \( x \), and (iii) each of its query children \( w \) is (a) (recursively) unconstrained and labeled with (NC), and (b) at least one path between \( x' \) and \( y' \) is non-optional. The SC annotation decision for an unconstrained node can be made without regard to other nodes in its query subtree. Finally, in a tree-pattern query, we say two nodes in a tree are cousins of each other if neither is an ancestor (proper or not) of the other. Note that cousins have a common ancestor that is different from them both. Also note that \( x \) and \( y \) may be cousins in a query but the corresponding DTD nodes \( x' \) and \( y' \) may have an ancestor-descendant relationship.

For any query nodes \( x \) and \( y \), suppose in any valid instance, either the security levels of the corresponding matches of \( x \) and \( y \) are the same or whenever they are not. This is solely because the security level at (the match of) \( x \) is locally changed. In this case, we say that the (match of) \( x \) has almost the same security level as the (corresponding match of) \( y \).

### 3.4 DAG-structured DTD Graphs

In this section, we discuss the optimization of SC annotations of arbitrary tree pattern queries, for the case of DAG-structured DTD graphs. Tree-structured DTDs have been studied in [28]. There is a considerable conceptual leap from tree-structured DTDs to DAG-structured DTDs, since there are multiple paths between pairs of nodes, each with a different sequence of labels. A straightforward extension of the reasoning over a single schema path for tree-structured DTDs to exponentially many schema paths for DAG DTDs would lead to an exponential-time algorithm for the case of DAG schemas. How then can we do such reasoning efficiently over DAG DTDs? Specifically, we consider the following questions: (1) When can we optimize a SC annotation of a query node when the context of the matching node in DTD gets more complicated due to multiple parents/ancestors? (2) With multiple query nodes, in what order should
we process the optimization on them to reach the minimal SC annotation? (3) Is the minimal correct SC annotation always unique? As a preview, we propose four rules for optimizing query nodes (Root, PC, CP, and CC), which are based on four different relationships of the query nodes. We show these rules to be sound and complete to optimize the SC annotations of query nodes. Based on these results, we develop a polynomial-time algorithm that is guaranteed to reach a minimal SC annotation for a given query. In this section, we provide the pseudo code for each rule and explain each separately. We use the DTD shown in Figure 3.2 for all examples in this section.

### 3.4.1 Root Rule

When the query root \( x \) is such that all paths from the DTD root to the corresponding node \( x' \) are non-overriding (defined in Section 3.3), we can optimize the SC annotation of \( x \): if \( x' \) is overriding, we can optimize the SC annotation of \( x \) to (LC), otherwise to (NC). As an example, in Figure 3.3(a), since the path (which is the only path) from the DTD root (OnlineSeller) to Accounts is non-overriding, and the DTD node Accounts is labelled with (M), the SC annotation of query root Accounts is optimized as (LC).
3.4.2 Parent-Child (PC) Rule

Let $x$ be a query node and $y$ be its query parent. Let $x'$, $y'$ be the corresponding DTD nodes. We can optimize the SC annotation on $x$ if any data node that matches $x$ has almost the same security level as the corresponding match of $y$. There are three possible scenarios for such optimization.

**Case I:** The query in Figure 3.3(a) involves a direct PC-edge. We can optimize the SC annotation of such a query node $x$ whenever the corresponding DTD node $x'$ is non-overriding. This scenario is true for node Account, and so its SC annotation is optimized to (LC) in Figure 3.3(a).

**Case II:** Consider an AD-edge $(y, x)$ in the query. If all DTD paths from $y'$ to $x'$ are non-overriding, i.e., there is no node to introduce a new security level between $x$ and $y$, then we can optimize node $x$. In the query shown in Figure 3.3(b), all DTD paths from Items to ID are non-overriding and ID is non-overriding, so the SC annotation of ID can be optimized to (NC). The pseudo code can be found in lines 1-4 in Figure 3.5.

**Case III:** Finally, let $(y, x)$ be an AD-edge and suppose $x$ is unconstrained (defined in Section 3.3). Then we can optimize the SC annotation of $x$ if a match of $y$ in any instance is guaranteed to have a descendant that matches $x$ with almost the same security level as the said match of $y$. In terms of DTD paths, it is enough to show that on every path $P$ from $y'$ to $x'$, there is a node $z'$ (possibly $z' = y'$), such that there is at least one non-optional non-overriding DTD path from $z'$ to $x'$ (see the definition of non-optional path and non-overriding path in Section 3.3), and the segment of the path $P$ from $y'$ to $z'$ is non-overriding. Note here if $z'$ is an intermediate node, $z'$ must be non-overriding. For example, for the nodes OnlineSeller and ID in the query shown in Figure 3.3(d), the path \{OnlineSeller, Items, Book, ID\} in the DTD (Figure 3.2) is such a path.\(^5\)

\(^5\) $z' = y'$ in this case.
This path tells us that any data node OnlineSeller is guaranteed to have a descendant with tag ID that has the same security level as itself. Therefore, we can optimize the SC annotation of ID to be (NC). The pseudo code can be found in lines 5-11 in Figure 3.5.

To summarize, we have three cases for optimizing a query node based on its query parent. The first two cases apply to both constrained and unconstrained nodes while the last applies only to unconstrained nodes. We have the following result:

**Lemma 3.4.1 (Completeness of PC rule):** Let \( Q \) be a tree-pattern query, \( \Delta \) be any disjunctionless DAG DTD, and \((y, x)\) be an (ancestor-descendant or parent-child) edge in \( Q \). Suppose \( D \) is a valid database instance of \( \Delta \). Then in every matching \( \mu : Q \rightarrow D \), \( \mu(x) \) and \( \mu(y) \) have almost the same security levels if and only if one of the three cases above holds.

Note that if one of the cases I-III holds for a query node \( x \) but the DTD node \( x' \) is overriding, then \( x \) can be optimized to (LC). The proof of Lemma 3.4.1 is shown in Appendix A.

### 3.4.3 Child-Parent (CP) Rule

The core intuition behind optimizing a query node using the CP rule is that it is guaranteed to have (almost) the same security level as the match of its query parent. CP rule is similar to PC rule. The only difference is that we optimize the query parent node instead of the query child whenever one of the cases I-III above holds. For example, in Figure 3.3(e), two correct SC annotations \( A_1, A_2 \) are shown. In both SC annotations, the root Accounts is optimized to (LC). In \( A_1 \), the SC annotation of node Item is optimized to (NC) using the PC rule (both case II and case III can be applied), whereas in \( A_2 \), the node PaymentInfo is optimized using the CP rule (only case III can be applied). For this example, \( A_1 \) and \( A_2 \) are both minimally correct SC annotations. Thus, the minimally correct solution to the security check annotation optimization problem is not unique!

### 3.4.4 Cousin-cousin (CC) Rule

When both PC and CP rules fail to apply, it is still possible to optimize a query node \( x \) by CC-rule. The intuition is that \( x \) may have a cousin \( y \) in the query tree such that \( y \) is annotated (RC) or (LC). From the DTD, we can infer that in any valid database instance, any match of \( x \) has almost the same security level as a corresponding match of \( y \). Verifying this condition based on the DTD graph is somewhat complex, so we break it down into two cases, as follows.

---

6 This means we do not count the label on \( x' \) when deciding if a path to \( x' \) is overriding.

7 Thus, since the security level of \( y \) is checked, there is no need to do a recursive check on \( x \).
3.4.4.1 Unconstrained Nodes

Suppose $x$ is unconstrained and $y$ is annotated. Let $w$ be the lowest common ancestor of $x$ and $y$ in the query tree and $S_x$ ($S_y$) be the set of ancestors of $x$ ($y$) occurring on the path from $w$ to $x$ ($y$) in the query tree, including $w$. Intuitively, $S_x$ ($S_y$) captures the relevant query context of node $x$ ($y$). Let $S_x'$ ($S_y'$) be the set of DTD nodes corresponding to $S_x$ ($S_y$). We wish to optimize $x$ whenever (a match of) $x$ must have almost the same security level as (a corresponding match of) $y$. There are three possible scenarios: $x'$ and $y'$ are cousins (case IV.1); $x'$ is the descendant of $y'$ (case IV.2); or $x'$ is the ancestor of $y'$ (case IV.3). The details of these cases are given below.

Case IV.1: Suppose there is at least one overriding node between $w'$ and $y'$, and that for every overriding node $z'$ between $w'$ and $y'$ that is closest to $y'$ within the context of $S_y'$, we see that there is a DTD path $P$ from $w'$ to $x'$ that passes through this overriding node $z'$ as well as through all nodes in $S_x'$, such that the segment of $P$ from $z'$ to $x'$ is non-optional and non-overriding (as illustrated in Figure 3.7(a)). This case guarantees that in any valid instance, for any match of $y$, there must exist a corresponding match of $x$ occurring in the right query context such that both matches inherit their security level from the same $w$. Therefore, if $y$ is annotated (RC), we can optimize $x$ to (LC) whenever $x'$ is overriding, and to (NC) otherwise. The pseudo code is shown on lines 11-16 in Figure 3.8. Using $Q_1$ in Figure 3.6 as an example (the DTD is shown in Figure 3.2), Item has two closest overriding ancestors: Cart and Account. Since for each of them there exists a non-overriding non-optional path to node ID, no matter from which node Item inherits the security level, we can ensure such a node must have an ID descendant that has the same security level as Item. Thus we put (NC) on ID while keeping (RC) on Item.

Case IV.2: The second scenario for optimizing $x$ is that $y'$ occurs on some path $P$ from $w'$ to $x'$, and this path passes through all nodes in $S_x'$, thus carrying the right query context of node $x$. Suppose for each $P$ within the query context of $S_y'$, the segment of $P$ from $y'$ to $x'$ is non-overriding and non-optional. As a result, for any match of $y$ we are guaranteed of an existence of a match of $x$ with almost the same security level as (the match of) $y$ (illustrated in Figure 3.7(b)). Thus, if $y$ is annotated (RC) or (LC), we can optimize the SC annotation of $x$ (to (NC) if $x$ is not overriding, and to (LC) otherwise). Using $Q_2$ in Figure 3.6 as an example, there exists a non-optional, non-overriding path from Account to ID within the query context, thus we put (NC) on ID while keeping (RC) on
Chapter 3. Query Evaluation over Access-controlled XML Databases

3.4.4.2 Constrained Nodes

When \( x \) is constrained, we must ensure that every one of its matches that satisfies the query conditions is accessible.

Let \( x \) be a query node and \( w \) be any query ancestor of \( x \). Let \( S \) be the set of query nodes appearing on the path from \( w \) to \( x \) in the query tree. If there are no overriding nodes between \( w' \) and \( x' \) in the DTD, then we could always apply the PC rule to optimize \( x \), so suppose that there are such nodes. We can optimize \( x \) provided that for every overriding node \( z' \) between \( w' \) and \( x' \) that is closest to \( x' \), such that \( z' \) is on a DTD path that starts at \( w' \), goes through the nodes \( S' \), and ends in \( x' \),\(^{8}\) we can find a cousin \( y \) of \( x \) in the query satisfying certain obligations. The first obligation is that \( y \) should be annotated (RC) since its security level must be explicitly checked. The second is that \( w \) must be the lowest common ancestor (LCA) of \( x \) and \( y \) in the query. The remaining obligation is that one of the following conditions discussed below, Case V.1 (\( x' \) and \( y' \) are cousins), Case V.2 (\( x' \) is the descendant of \( y' \)), and Case V.3 (\( x' \) is the ancestor of \( y' \)), must hold. These conditions are similar to those discussed for

---

\(^{8}\)This path captures the correct query context for node \( x \).
Case IV for the unconstrained case. Let \( T \) be the set of query nodes appearing on the path from \( w \) to \( y \) in the query tree.

**Case V.1:** Suppose \( z' \) is a common ancestor of \( x' \) and \( y' \) in the DTD, that there is exactly one path \( P \) from \( w' \) to \( y' \), and that the segment \( P_1 \) of \( P \) from \( w' \) to \( z' \) is unique (see the definition of unique path in Section 3.3). Furthermore, suppose \( P \) goes through all nodes in \( T' \), thus carrying the right query context for \( y \). In addition, suppose there are no overriding nodes between \( z' \) and \( y' \) nor between \( z' \) and \( x' \) (illustrated in Figure 3.10(a)). Under these conditions, for any match of \( y \), we are assured that every corresponding match of \( x \) inherits its security level from a match of \( w \) from which \( y \) inherits its security level as well. One or more matches of \( x \) may override this locally, if at all. It follows then that \( x \) can be optimized to (LC) if \( x' \) is overriding, and to (NC) otherwise. Using Figure 3.9(a) as an example, there exists only one DTD path from Customer to Date, on which the DTD node Cart is overriding. Since the DTD path from Customer to Cart is unique, there only exists one instantiated node of Cart under Customer. Note that since Cart is labeled with (0), it may be assigned
a different security level from Customer. However, since there is no overriding node between either Cart and Date nor Cart and Item, we put (NC) on Item while keeping (RC) on Date, i.e., for every accessible Date, the security level must be inherited from the only accessible source Cart, which is also true for Item. It is obvious that if the query root is changed to Account (Figure 3.9(b)), the SC annotation on both Date and Item cannot be optimized. This is because the DTD path from Account to Cart is not unique! The pseudo code is given by lines 12-16 in Figure 3.11.

Case V.2: Suppose \( y' \) is an ancestor of \( x' \) in the DTD and \( z' \) is an ancestor of \( y' \). Furthermore, suppose there is only one DTD path \( P \) from \( w' \) (the corresponding instance node of the lowest common ancestor \( w \) of \( x \) and \( y \) in the query) to \( y' \) that passes through the query context nodes \( T' \), such that the segment \( P_1 \) of \( P \) from \( w' \) to \( z' \) is unique, and that there are no overriding nodes between \( z' \) and \( x' \) (illustrated in Figure 3.10(b)). In this case, in any valid instance, for any match of \( y \), every corresponding match of \( x \) has almost the same security level as that of (the corresponding match of) \( z \), which is also the security level of (the match of) \( y \), and both matches appear in the right query context, so \( x \) can be optimized to (LC) if \( x' \) is overriding and to (NC) otherwise. Using Figure 3.9(c) as an example, there exists only one DTD path from Customer to Cart, which is also unique. Since there is no overriding node between either Cart and Date, we put (NC) on Date while keeping (LC) on Cart, i.e., there exists only one instantiated node Cart below Customer, from which node Date will inherit the security level. Thus, if such a Cart is accessible, all Date nodes below Customer must be accessible as well. The pseudo code is given by lines 17-20 in Figure 3.11.
Function case_V(query node x, x’s query cousin list Clist)
1 If 3 ancestor w of x in query {
2 Let S be the set of ancestors of x occurring on the path from w to x in query;
3 y or y’ closest to x’ within the context of S
4 y’ in Clist in query s.t.
5 y = w or y below w in query AND w is LCA of x and y AND
6 case.W1(w, x, y, o, T) != NULL OR case.W2(w, x, y, o, T) != NULL
7 OR case.W3(w, x, y, o, T) != NULL
8 //T: the set of ancestors of y occurring on the path from w to y in query
9 If (only case.W3 applied) return ‘‘NC’’;
10 Else return SimplifyBelow(x);
11 Else return SimplifyAbove(x);
END Function

Function case.W1(query node x, x’s query cousin y, query LCA w, overriding node o,
y’s query context T)
12 If y’ is not descendant or ancestor of x’ in DTD AND only one
13 path P from w’ to y’ within T AND forbidden(y’) AND o’ is ancestor
14 of y’ in DTD on P AND unique(w’, o’) on P AND ornodeset(o’, y’) = NULL on P
15 Return y;
16 Else Return NULL;
END Function

Function case.W2(query LCA w, query node x, x’s query cousin y, overriding node o,
y’s query context T)
17 If only one path P from w’ to y’ within T AND y’ is ancestor of x’ on P AND
18 ornodeset(y’, x’) == NULL on P AND o’ is ancestor of y’ on P AND unique(w’, o’) on P
19 Return y;
20 Else Return NULL;
END Function

Function case.W3(query LCA w, query node x, x’s query cousin y, overriding node o,
y’s query context T)
21 If only one path P (w’, y’) within T AND y’ is descendant of x’ on P AND forbidden(y’)
22 AND ancestor (o’, x’) on P AND ornodeset(x’, y’) == NULL on P AND unique(w’, o’) on P
23 Return y;
24 Else Return NULL;
END Function

Figure 3.11: CC Rule for Constrained Nodes

Case V.3: Finally, suppose x’ is an ancestor of y’ in the DTD and z’ is an
ancestor of x’. Furthermore, suppose there is exactly one path P from w’ to
y’ that passes through the context nodes T’, and that the segment P1 of P
from w’ to z’ is unique (illustrated in Figure 3.10(c)). Suppose also that there
are no overriding nodes between z’ and y’. Then, in any valid instance, every
match of y must be a descendant of a corresponding match of x. Since there
are no overriding nodes between them, they must have the same security level,
allowing us to optimize x. Just as for Case IV.3, we can optimize x to (NC), even
if x’ is overriding. The reason is that the check performed for y subsumes the
necessary check for x. Figure 3.9(d) is an example in which the SC annotations
are optimized by applying this rule. One thing to note from this query is that
Cart is labeled as (NC), although it is labeled as (0) in the DTD. The pseudo
code is given by lines 21-24 in Figure 3.11.

We derive the following results:
Lemma 3.4.2 (Completeness of CC – Unconstrained): Let $Q$ be a tree-pattern query, $\Delta$ be a DTD, and $x$ be an unconstrained node in $Q$. Suppose $x$ cannot be optimized using PC and CP rules. Then $x$ can be optimized if and only if one of the conditions Case IV or Case V holds.

Lemma 3.4.3 (Completeness of CC – Constrained): Let $Q$ be a tree-pattern query, $\Delta$ be a DTD, and $x$ be a constrained node in $Q$. Suppose $x$ cannot be optimized using PC and CP rules. Then $x$ can be optimized if and only if Case V holds.

The proofs of Lemma 3.4.2 and Lemma 3.4.3 are shown in Appendix A.

3.4.5 Putting It All Together

We can show that the rules Root, PC, CP, CC are complete for optimizing nodes in a tree-pattern query tree. We start with Lemma 3.4.4.

Lemma 3.4.4 (Redundancy of Reasoning Query Ancestor/Descendant): Let $Q$ be a tree-pattern query, $\Delta$ be any disjunctionless DAG DTD, $x$ be an arbitrary query node in $Q$, $y$ be its query parent, and $z$ be an query ancestor of $x$. Suppose $D$ is a valid database instance of $\Delta$. Then in every matching $\mu: Q \rightarrow D$, if $\mu(x)$ and $\mu(z)$ have almost the same security levels, $\mu(y)$ must have the same security levels.

The correctness of Lemma 3.4.4 is obvious: if the instantiated node $x'$ of the query node $x$ always has almost the same security level as the instantiated node $z'$ of $x$'s query ancestor $z$, there should be no node between $x'$ and $z'$ that brings in a new security level, i.e., any instantiated node $y'$ of $x$'s query parent $y$ is guaranteed to have the same security level as $x'$.

Lemma 3.4.4 shows that if the SC annotation of any query node can be optimized based on the reasoning on its query ancestor/descendant, it should be optimized by reasoning on its query parent/child. We can then state the following theorem.

Theorem 3.4.1 (Completeness of Rules): Let $Q$ be a tree-pattern query with all its nodes annotated (RC), and $\Delta$ be any disjunctionless DAG DTD. Then the SC annotation on a node $x$ in $Q$ is optimizable if and only if it can be optimized using one of the rules \{Root, PC, CP, CC\}.

Proof:

The essence of the optimization of the SC annotation of a query node $p$ is the existence of a “witness” in the query to ensure $p$'s accessibility. Such a “witness” can be any query node other than itself. Thus it must be either its query parent, child, cousin, ancestor or descendant. For the query root, the Root rule covers the optimization. For the non-root query nodes, since the reasoning on both ancestor and descendant is redundant (Lemma 3.4.4), the reasoning on the query parent (PC rule), child (CP rule), and cousin (CC rule) is enough.
Chapter 3. Query Evaluation over Access-controlled XML Databases

Unfortunately, Theorem 3.4.1 does not yield an efficient algorithm for the security-check annotation optimization problem. It is not clear in what order the rules should be applied and in what order query nodes should be visited. We can show the following lemmas concerning rule applications. For convenience, we use regular expressions to denote the order of rule applications. For example, Root(PC|CC|CP)* indicates applying Root first, followed by any sequence of rule applications – PC, CC, CP. For two SC annotations \( A \) and \( B \) of a query, we write \( A <_X B \), where \( X \in \{ \text{RC, LC, NC} \} \), provided the number of query nodes mapped to \( X \) by \( A \) is less than the number of nodes mapped to \( X \) by \( B \). The relation \( =_X \) is defined similarly. For example, \( A <_{\text{RC}} B \) means that \( A \) annotates fewer nodes with an (RC) than \( B \).

Lemma 3.4.5 (Remove CP(s)) : Let \( Q \) be any query, \( \Delta \) be any disjunctionless DAG DTD, and \( A \) be the SC annotation assigning (RC) to all nodes of \( Q \). For any (correct) SC annotation \( A' \) obtained from \( A \) by applying \( \{ \text{Root, PC, CP, CC} \}^* \), we can always reach a correct SC annotation \( B \) by applying \( \{ \text{Root, PC, CC} \}^* \). We have either (i) \( B <_{\text{RC}} A' \) or (ii) \( B =_{\text{RC}} A' \) and \( B <_{\text{LC}} A' \).

For example, in Figure 3.3(e), SC annotation \( A_1 \) would be obtained using Root(PC*)(CC*), whereas \( A_2 \) would be obtained using Root(CP*). Clearly, \( A_1 =_{\text{RC}} A_2 \) and \( A_1 =_{\text{LC}} A_2 \).

Lemma 3.4.6 (Move PC(s) Ahead) : Let \( Q \) be any query, \( \Delta \) be any disjunctionless DAG DTD, and \( A \) be the SC annotation assigning (RC) to all nodes of \( Q \). For any (correct) SC annotation \( A' \) obtained from \( A \) by applying \( \{ \text{Root, PC, CC} \}^* \), we can always reach a correct SC annotation \( B \) by firstly repeatedly applying the PC rule until the SC annotations of no more nodes can be optimized, then applying CC rules in the same fashion. We have either (i) \( B <_{\text{RC}} A' \) or (ii) \( B =_{\text{RC}} A' \) and \( B <_{\text{LC}} A' \).

The correctness of Lemmas 3.4.5 and 3.4.6 are based on the study of the defined Root, PC and CC rules. More details of the proof are provided in Appendix A. Based on Lemmas 3.4.5 and 3.4.6, we can show the following "normal form" result:

Theorem 3.4.2 (Order of Optimization Rules) : Let \( Q \) be any query, \( \Delta \) be any disjunctionless DAG DTD, and \( A \) be the SC annotation assigning (RC) to all nodes of \( Q \). Then the SC annotation \( B \) obtained by applying Root(PC*)(CC*) to query nodes, with the PC* and CC* closures applied top-down, is correct and minimal. Furthermore, for any (correct) SC annotation \( A' \) obtained from \( A \) by applying \( \{ \text{Root, PC, CP, CC} \}^* \), we have either (i) \( B <_{\text{RC}} A' \) or (ii) \( B =_{\text{RC}} A' \) and \( B <_{\text{LC}} A' \).

In particular, the theorem shows that any SC annotation we could have obtained is equaled or bettered by applying rules according to the normal form above. The proof can be found in Appendix A.

We can show:
Chapter 3. Query Evaluation over Access-controlled XML Databases

Theorem 3.4.3 (Complexity): Let $Q$ be any query and $\Delta$ be any disjunctionless DAG DTD. Then a minimal correct SC annotation of $Q$ with regard to $\Delta$ can be obtained in polynomial time in the size of $Q$ and $\Delta$. 

Proof: The main intuition behind polynomiality is that Root, PC and CC rules base their reasoning on overriding nodes (referred to in the pseudo code as ornodeset), using which it captures all overriding paths that could potentially affect the optimality of a query node. For each such node, it checks the existence of certain paths in the DTD. Additionally, for constrained nodes, it checks the absence of certain paths.

For these, certain paths that will be checked are listed as follows:

- We say a node $z'$ is on every path from $x'$ to $y'$, $\text{OEP}(x', y', z')$, if this holds in a conventional graph theoretical sense; this check can be done in polynomial time by using depth first search.

- There is a non-optional non-overriding path from $x'$ to $y'$, $\text{NONOR}(x', y')$, if and only if there is a path from $x'$ to $y'$ such that all edges in the path are labeled '1' or '1' and all intermediate nodes are non-overriding; this check can be done in polynomial time by the following procedure: there is a non-overriding non-optional path from DTD node $x'$ to $y'$ if and only if:
  (a) nodes $y'$ and $x'$ are identical; or (b) there exists a non-overriding child $e'$ of $x'$ such that the edge$(x', e')$ is labeled with '1' or '1' and $\text{NONOR}(e', y')$.

- There is a unique path from $x'$ to $y'$, $\text{UP}(x', y')$, if and only if there is a path from $x'$ to $y'$ such that all edges in the path are labeled '?' or '1'; this check can be done in polynomial time using the following procedure: there is a unique path from DTD node $x'$ to $y'$ if and only if: (a) nodes $y'$ and $x'$ are identical; or (b) there exists a child $e'$ of $x'$ such that the edge$(x', e')$ is labeled with '?' or '1' and $\text{UP}(e', y')$.

- There is only one path from $x'$ to $y'$, $\text{OOP}(x', y')$, if and only if there is exactly one path from $x'$ to $y'$. This check can be done in polynomial time using the following procedure: there is only one path from DTD node $x'$ to $y'$ if and only if: (a) nodes $y'$ and $x'$ are identical; or (b) there exists only one child $e'$ of $x'$ that has path to $y'$ and $\text{OOP}(e', y')$. The "only one child" condition is controlled by adding a counter when using recursion.

To sum up, since checking of Root, PC and CC rules can be implemented polynomially, a minimally correct SC annotation of query $Q$ with regard to a DAG-structured DTD $\Delta$ can be obtained in polynomial time in the size of $Q$ and $\Delta$. 

### 3.5 DAG-Structured DTDs with Disjunction

A DAG DTD $\Delta$ with disjunction nodes can be unfolded into a set of disjunctionless DTDs by choosing one of the sequences from each disjunction production. Each set of disjunctions results in a disjunctionless DTD. We call the set of such DTDs candidate DTDs of $\Delta$. For example, the disjunction DTD in Figure 3.12 (a) has two candidate DTDs: $\Delta_1 = \{A \rightarrow (B?C^*)?, B \rightarrow (EF^*)\}$ (Figure 3.12 (b)), and $\Delta_2 = \{A \rightarrow (CD^*)\}$ (Figure 3.12 (c)). It is straightforward to demonstrate the following proposition:

**Proposition 3.5.1 (Candidate DTDs):** Let $Q$ be a tree-pattern query, $\Delta$ be a disjunction DTD, and $\text{cand}(\Delta)$ be its associated set of candidate DTDs. Then a SC annotation on $Q$ is correct with regard to $\Delta$ if and only if it is correct with regard to every DTD in $\text{cand}(\Delta)$.

### 3.5.1 The Challenges

Unfortunately, the above result doesn’t yield an efficient algorithm for optimizing the SC annotations of a query with regard to disjunction DTDs, since a disjunction DTD can have exponentially many candidate DTDs. For example, the DTD in Example 3.5.1, graphed in Figure 3.13, has $2^2 = 4$ candidate DTDs.

**Example 3.5.1 [Disjunction DTD]**

We will use the DTD $\Delta = \{A \rightarrow B \mid C, B \rightarrow D^*H, C \rightarrow D, D \rightarrow E \mid F, E \rightarrow G, F \rightarrow G, G \rightarrow H^*I^*\}$ as the running example throughout this section. The DTD graph is shown (in simplified form, per Section 2.1.1) in Figure 3.13. In this DTD, elements $A$ and $F$ are marked (M), and $D$ and $E$ are marked (O), all other elements being (F).

The following example shows there is no obvious connection between optimality of query nodes against disjunctionless DTDs and a corresponding disjunction DTD.
Example 3.5.2 [Disjunctionless vs. Disjunction DTDs]
Consider the DTD $\Delta_1 = \{A\rightarrow BC, B\rightarrow D, C\rightarrow D\}$, with $C$ being optional (O). In the query $A[.//D]$, $D$ can be optimized since there is a non-overriding non-optional path from $A$ to $D$ (Case III). If we change the first production to $A\rightarrow B | C$, we cannot optimize $D$, since the optimization would not be correct with regard to the candidate DTD $\{A\rightarrow C, C\rightarrow D\}$.

Now, consider the disjunction DTD $\Delta_2 = \{A\rightarrow B | C, B\rightarrow D*E*, C\rightarrow DE*\}$, with $C$ being optional (O). In the query $A[.//D][.//E]$, we can optimize $D$, since in one candidate DTD, we can apply PC (Case II) and in the other we can apply CC (Case IV.1) successfully. However, if we remove the disjunction nodes from the DTD by changing the first production to $A\rightarrow BC$, then $D$ can no longer be optimized. Figure 3.14 shows an instance demonstrating that $D$ must be annotated (RC).

Since there is no connection between optimality of query nodes against disjunctionless DTDs and that with regard to a corresponding disjunction DTD, we need a clever strategy for optimizing the SC annotations against disjunction DTDs.
3.5.2 Optimizing against Disjunction DTDs

An important and surprising fact is that we can show that even for disjunction DTDs, we can give a small set of sound and complete rules—Root, PC, CP, and CC. These rules need to be suitably redefined to handle disjunction. The key challenge is the existence of an exponential number of candidate DTDs underlying a disjunction DTD, where a query node may be optimizable in different candidate DTDs for different reasons. To keep track of these candidates, we introduce a labeling scheme that assigns a label to each node.

Our labeling scheme permits us to keep track of the membership of DTD nodes in candidate DTDs with a polynomial amount of information. Specifically, we assign each disjunction node in the DTD a unique integer. We also assign each sequence node out of the disjunction node a distinct integer. Labels are sets of entries of the form \( i.j \) and are computed using the following rules:

1. Every ancestor of a disjunction node \( i \) (including node \( i \)) has a label that includes all entries of the form \( i.j \), for every sequence node \( j \) out of node \( i \).
2. For a disjunction node \( i \) and a sequence node \( j \) out of node \( i \), node \( j \) inherits the label of node \( i \), except for all entries in the inherited label of the form \( i.k \), where \( k \neq j \), are removed.
3. Whenever \( y \) is an element node, it inherits the label of every sequence node from which there is an incoming edge into \( y \) in the DTD graph.

Any label assignment to nodes of a disjunction DTD, satisfying the above rules, is a (valid) labeling. Labeling for a (DAG) disjunction DTD can be computed efficiently with a backward propagation pass followed by a forward propagation. Figure 3.15(a) illustrates the labeling scheme. Note that labeling takes up space that is linearly bounded by the DTD size.

**Definition 1 [Labeling]** Let \( \Delta \) be a disjunction DTD, \( \Delta_c \in cand(\Delta) \) be a candidate DTD, and \( L \) be any label. Then \( \Delta_c \) is covered by \( L \), provided that for every disjunction node \( i \) in \( \Delta \), whenever sequence node \( j \) out of \( i \) is present in \( \Delta_c \), the entry \( i.j \) is present in \( L \).

For example, in Figure 3.15(a), the label \( L = (1.1, 1.2, 2.1) \) covers the candidate DTD \( \{A\rightarrow B, B\rightarrow D*H, C\rightarrow D, D\rightarrow E, E\rightarrow G, F\rightarrow G, G\rightarrow H*1\} \). The following result is straightforward.

**Proposition 3.5.2 (Labeling):** Every disjunction DAG DTD \( \Delta \) has a unique labeling. For any node \( x \) in \( \Delta \), a candidate DTD \( \Delta_c \) of \( \Delta \) contains node \( x \) if and only if the label of \( x \) covers \( \Delta_c \).

For example, in Figure 3.15(a), the label \( L = (1.1, 1.2, 2.1) \) of node \( E \) covers the candidate DTD above and is present in it.

\[ ^9 \text{Note that the figure shows a simplified DTD graph, so some sequence/disjunction nodes are implicit.} \]
3.5.3 Modified Rules

We now redefine the four rules, Root, PC, CP, and CC (defined in Section 3.4), adapted for handling disjunction nodes in the schema. As a quick summary, the major change is that rule applications also have to manipulate and update labels to reflect the reasoning employed. Additionally, some of the reasoning about paths needs to be universal, to allow for disjunction. Below, we assume \( x \) is the node to be optimized. We denote the label of a DTD node \( x' \) by \( L_{x'} \).

**Root:** This rule is identical to that for disjunctionless DTDs.

**PC:** Let \( y \) be the query parent of \( x \). Recall Cases I-III defined for disjunctionless DTDs. Of these, Case I, applicable when the query edge \((y, x)\) is a PC-edge (parent-child edge), is unchanged. Disjunction nodes cannot affect such direct edges. So, suppose \((y, x)\) is an AD-edge (ancestor-descendant edge). Case II applies when there are no overriding DTD nodes between \( y' \) and \( x' \), and is also unchanged. The intuition is that there are no intervening overriding nodes in any candidate DTD. Case III is similar. Suppose there are overriding nodes between \( y' \) and \( x' \). For every such node \( o' \) that is closest to \( y' \), for every ancestor \( w' \) of \( o' \) that is between \( y' \) and \( o' \), we test to see if there is a non-overriding non-optional path from \( y' \) to \( w' \). If so, we subtract \( L_{w'} \) from \( L_{o'} \). At the end, if for every overriding node \( o' \) between \( x' \) and \( y' \) that is closest to \( y' \), the label \( L_{o'} \) is empty, we can optimize \( x \). The intuition is that in every candidate DTD where \( o' \) is present, there is a non-overriding non-optional path that acts as a "bypass". Here, a critical point is the test for a non-overriding non-optional path: it needs to be interpreted in a universal sense. More precisely, we derive the following definition:

**Definition 2 [Non-overriding non-optional path]** There is a non-overriding non-optional path from DTD node \( x' \) to \( y' \), \( \text{NONOR}(x', y') \), if and only if: (i) \( x' \) is an element node, its disjunction child is \( c' \) and \( \text{NONOR}(c', y') \) holds; or (ii) \( x' \) is a disjunction node and for each sequence child \( s' \) of \( x' \), \( \text{NONOR}(s', y') \) holds;
or (iii) $x'$ is a sequence node and there exists an element child $e'$ of $x'$ such that the DTD label on the edge $(x', e')$ is '1' or '+' and $\text{NONOR}(e', y')$; or (iv) nodes $x'$ and $y'$ are identical.

We can test the $\text{NONOR}$ predicate efficiently using depth-first search. Note that in addition to subtracting $L_w$ from $L_{o'}$ above, we could "propagate" this change by subtracting $L_{o'}$ from the label of every overriding node that is between $o'$ and $x'$. We assume that this is the case in the following rules.

**CP:** As with disjunctionless DTDs, CP is symmetric to PC: it uses Cases I-III but to optimize a node based on its query child. We omit the details.

**CC:** The key differences from the PC rule are that (i) applying the CC rule (both for constrained and unconstrained nodes) also entails label manipulation, and (ii) reasoning about the existence of a "unique path", about "only one path" between a pair of DTD nodes, and about a node being present in "every path" between two other nodes, needs to be done in the context of labels (which capture candidate DTDs). We highlight these changes next.

A disjunctionless DTD has the following properties: (i) we say a node $z'$ is on every path from $x'$ to $y'$, $\text{OEP}(x', y', z')$, if this holds in a conventional graph theoretical sense; (ii) there is a unique path from $x'$ to $y'$, $\text{UP}(x', y')$, if and only if there is a path from $x'$ to $y'$ such that all edges in the path are labeled '?' or '1'; and (iii) there is only one path from $x'$ to $y'$, $\text{OOP}(x', y')$, if and only if there is exactly one path from $x'$ to $y'$.

In a disjunction DTD, $\text{OEP}_{ch}(x', y', z')$, $\text{UP}_{ch}(x', y')$, and $\text{OOP}_{ch}(x', y')$ are functions that return a label $L$ such that in any candidate DTD $\Delta$, the corresponding predicate for disjunctionless DTD holds if and only if $\Delta$ is covered by $L$. Each of these functions can be implemented efficiently.

Finally, the CC rule itself is applied as follows. As with disjunctionless DTDs, if the node $x$ that is being optimized is a constrained node, only Case V (see below) applies, otherwise both Cases IV and V apply. Case IV for disjunction DTDs is defined to be Case IV for disjunctionless DTDs, except that the new definitions of $\text{NONOR}_{ch}$ and $\text{OEP}_{ch}$ given above are used in place of the corresponding definitions of $\text{NONOR}$ and $\text{OEP}$. Similarly, Case V for disjunction is essentially the same as the disjunctionless schema, except that the new definitions of $\text{UP}_{ch}$, $\text{OOP}_{ch}$ are used in place of the corresponding definitions of $\text{UP}$, $\text{OOP}$. The upshot is that Case IV for disjunction can be regarded as a function that returns a label $L$ such that Case IV for a disjunctionless schema holds in a candidate DTD if and only if $\Delta$ is covered by $L$; The same holds for Case V. We apply these cases as follows.

For every query ancestor $w$ of $x$, for every overriding node $o'$ between $w'$ and $x'$ that is closest to $x'$ in the DTD, and for every cousin $y$ of $x$ in the query tree, apply Case IV for the 4-tuple of nodes $w', o', x', y'$ to obtain a label $L$. Then subtract $L$ from the label of $o'$ as well as from those of all overriding nodes between $w'$ and $o'$. The subtraction amounts to checking off candidate DTDs where $x$ is optimizable using Case IV. Similarly, we apply Case V on the nodes $w', o', x', y'$ to obtain a label $L$. Then subtract $L$ from the label of $o'$ as well as from those of all overriding nodes between $w'$ and $o'$; this procedure has an
analogous rationale. Finally, if all overriding nodes between \( w' \) and \( x' \) have an empty label at the end, we optimize \( x \) and exit; otherwise, we try the next query ancestor \( w \) of \( x \). Here, a small technicality is that whenever only Case IV.3 or Case V.3 is used to render labels of all overriding nodes between an ancestor \( w' \) and \( x' \) to be empty sets, then we can optimize \( x \) to (NC), unconditionally, exactly as we could do for disjunctionless DTDs. In other cases, we optimize \( x \) to (LC) if \( x' \) is overriding and to (NC) otherwise.

### 3.5.4 Algorithm DisjunctionOpt

The optimization algorithm for disjunction DTDs, Algorithm DisjunctionOpt, proceeds by applying the Root rule. Then it applies each of the PC and CC rules to saturation, in that order, again exactly like the algorithm for disjunctionless DTDs. We discuss this algorithm next.

**Example 3.5.3 [Illustrating Algorithm DisjunctionOpt]**

Consider the query in Figure 3.15(b). Initially let all nodes be annotated (RC). The Root rule is the same as for disjunctionless DTDs. So, first the query root \( A \) can be optimized to (LC) since the corresponding DTD node is marked (M).

Next, consider the query node \( H \). Let us apply the PC rule. Consider overriding DTD nodes between \( A \) and \( H \) that are closest to \( A \). \( D \) is the only such node. \( D \) has an ancestor \( B \), from which there is a non-overriding non-optional path to \( H \). Therefore, we subtract \( L_B \) from \( L_D \) as well as from \( L_E \) and \( L_F \) as part of change propagation (see Figure 3.15(c)). The label \( L_D = (1.2) \) is non-empty, signifying that in every candidate DTD that contains node \( C \) corresponding to sequence node 2 out of disjunction node 1, the PC rule (for disjunctionless DTDs) fails, meaning we cannot optimize \( H \) at this point. (If hypothetically, the DTD contained a non-overriding non-optional path from \( C \) to \( H \), then \( L_D \) would become empty and hence \( H \) could be optimized.)

Next, consider the CC rule on \( H \) and consider Case IV. The only query cousin of \( H \) is node \( I \) and its counterpart in the DTD is neither an ancestor nor a descendant of \( H \) in the DTD. Thus, Case IV.1 is the only subcase under Case IV that would apply. The only query ancestor of \( H \) is \( A \). The overriding DTD nodes between \( A \) and \( H \) that are closest to \( H \) are \( E \) and \( F \). But Case IV.1 fails for the 4-tuple of DTD nodes \( A, E, H, I \), since there is no non-optional non-overriding path from \( E \) to \( H \). Similarly, it also fails for \( A, F, H, I \).

Let us now apply Case V. Consider Case V.1. Node \( I \) corresponding to the query cousin \( I \) of \( H \) is neither an ancestor nor a descendant of \( H \) in the DTD. First, consider the overriding node \( E \). There is a unique path from \( A \) to \( E \), namely, \( A \rightarrow C \rightarrow D \rightarrow E \). The function \( \cup \mathbf{P}_{CH}(A,E) \) returns the label \( L_1 = (1.2,2.1) \), which corresponds to the intersection of the labels of nodes on this path. The function \( \cap \mathbf{P}(E,I) \) returns the label \( L_2 = (1.1,1.2,2.1,2.2) \), since such a path exists in every candidate DTD. There is a non-overriding path from \( E \) to \( H \) as well as from \( E \) to \( I \). Finally, the label \( L \) for this round of Case V is computed as the intersection \( L = L_1 \cap L_2 \). Then subtract \( L \) from the label
$L_E$ as well as from the label of every overriding node between $A$ and $E$. The result is shown in Figure 3.15(c). Note that $L_E$ becomes empty.

It is easy to check that when Case V.1 is applied for the overriding node $F$, we similarly end up emptying its label (see Figure 3.15(c)). Finally, $H$ can be optimized to (NC). Indeed, the SC annotation $\{A : (LC), H : (NC), I : (RC)\}$ is the optimal annotation for the query. The reader is invited to confirm that both the PC rule and the CC rule fail for $I$, even if $I$ is considered before $H$ for optimization.

### 3.5.5 Theoretical Results of Disjunction DTDs

We propose the following theorems for optimizing the SC annotations of a given query against disjunction DTDs. The first theorem shows that the modified rules are sound and complete.

**Theorem 3.5.1 (Completeness) :** Let $\Delta$ be a disjunction DTD and $Q$ be a query. The SC annotation on a node $x$ in $Q$ is optimizable if and only if it can be optimized using one of the modified Root, PC, CP or CC rules.

The next theorem shows that it is sufficient to apply rules in a certain order, analogously to disjunctionless DTDs.

**Theorem 3.5.2 (Order of Optimization Rules) :** Let $\Delta$ be a disjunction DTD and $Q$ be a query. Let $A$ be the SC annotation assigning (RC) to all nodes of $Q$, and $A'$ be any SC annotation obtained from $A$ by applying the rules {Root, PC, CP, CC} in any order. Then there is a SC annotation $B$ such that $B$ is correct, $B \leq A'$. $B$ can be obtained from $A$ by applying the Root rule, then the PC rule repeatedly to saturation to nodes of $Q$ top-down, followed by the CC rule repeatedly to saturation to nodes of $Q$ top-down.

Finally, we propose our optimality theorem:

**Theorem 3.5.3 (Optimality) :** Let $\Delta$ be a disjunction DTD and $Q$ be a query. Then a minimally correct SC annotation of $Q$ can be obtained in time polynomial in the size of $Q$ and $\Delta$.

The proof of Theorems 3.5.1 - 3.5.3 is provided in Appendix A.

### 3.6 Experiments

We ran an extensive set of experiments to measure the utility and effectiveness of security check optimization on various benchmark data sets and DTDs. Our experiments complement our analytical results, and demonstrate that security check minimization can make an order of magnitude difference in query evaluation performance, while incurring negligible overhead for optimization. We showed those results in this section.
3.6.1 Experimental Setup

We ran our experiments on the XMark XML benchmark dataset [111], a Biomedical dataset [98] from the National Biomedical Research Foundation, and the U.S. Congress Bills dataset [107] from the U.S. House of Representatives. We constructed two different security specifications for each DTD: (1) a DTD with only three elements marked (M) (we call this a sparse DTD); and (2) a DTD with one-quarter of its elements marked (M) and one-quarter of its elements marked (0) (we call this a dense DTD). We modified a few elements in the DTDs to eliminate cycles.

For each dataset, we constructed two kinds of XML documents, one on the sparse and one on the dense DTD, using IBM XMLGenerator [102]. For every DTD element marked (M), we assigned a value of 0 or 1 with equal probability to the SecurityLevel attribute of the corresponding instance nodes, 0 meaning “inaccessible” and 1 meaning “accessible”. For each element marked (0), we used a random number generator to decide whether to assign a value to the SecurityLevel attribute of the corresponding instance nodes. If the generator generated an even number, then a value of 0 or 1 is assigned to the SecurityLevel attribute of the corresponding instance nodes. Otherwise the SecurityLevel attribute of the corresponding instance nodes is left as unspecified.

We used the XALAN [110] engine for query evaluation, with external function calls for security checks. The optimization algorithms were implemented in Java. Our experiments were run on a Windows PC with 512MB of memory and 2GHz CPU speed. All reported results are the average of seven trials after the maximum and minimum were dropped.

3.6.2 Impact of Optimization

We used 10 queries for each dataset. These queries included PC-edges and AD-edges; some were path queries and the others were branching (tree-pattern) queries. Among branching queries, some branched only at the root, while others also branched below the root. From these 30 queries, we picked the queries in Figure 3.16 as examples to explain the experiment results, because the optimization on those four queries covered most of the optimization rules discussed in the previous sections. Query Q_1 is based on auction.dtd, Q_2 on protein.dtd,


<table>
<thead>
<tr>
<th>DTD</th>
<th>Type</th>
<th>Overriding Nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>XMark</td>
<td>sparse</td>
<td>site, payment, people</td>
</tr>
<tr>
<td>XMark</td>
<td>dense</td>
<td>site, payment, people, regions, parlist</td>
</tr>
<tr>
<td>Biomedical</td>
<td>sparse</td>
<td>ProteinDatabase, db, header</td>
</tr>
<tr>
<td>Biomedical</td>
<td>dense</td>
<td>ProteinDatabase, gene, ProteinEntry, xrefs</td>
</tr>
<tr>
<td>Congress</td>
<td>sparse</td>
<td>bill, legis-body, formula</td>
</tr>
<tr>
<td>Congress</td>
<td>dense</td>
<td>bill, formula, legis-body, non-statutory-material</td>
</tr>
</tbody>
</table>

Figure 3.17: DTDs w.r.t setup of overriding nodes

and $Q_3$ and $Q_4$ on bills.dtd. Figure 3.17 shows the overriding nodes within the query context we used for each DTD. Figure 3.18 shows the optimized SC annotation we obtained for each query.

We varied the size of datasets from 1MB to 100MB. The size of the secure answer is clearly the same for optimized query and unoptimized query.

Figure 3.19 shows the query evaluation time on XMark, Biomedical and Congress datasets. First, note that the benefits of our optimization on all datasets are substantial. For example, for the 100MB XMark dataset, the unoptimized secure evaluation on $Q_1$ for the sparse DTD is 20 seconds, whereas the optimized secure evaluation took only 6 seconds, a saving of 70%. Second, the benefit of the optimization on the sparse DTD is greater than on the dense DTD, since for the evaluation of the RC annotation (recall that it requires that the instance tree needs to be traversed upward recursively until an ancestor node that has a security level explicitly defined is reached), the recursive check needs to traverse more nodes, on average, to reach a node that has a specified security level when the data is sparsely annotated. Thus, the savings by optimizing the (RC) annotation to (LC) (i.e., a local check) or (NC) (i.e., a no-op) gains more savings on sparse DTD than on dense DTD. There are more such nodes that are optimized as LC or NC when there are fewer overriding nodes in the schema for sparse DTD than for dense DTD. The results of other queries were similar.

Figure 3.20 reports the query evaluation time for various percentages of instance elements whose corresponding DTD nodes are assigned (0) so that their SecurityLevel attributes in the instance were assigned a value by random. We applied unoptimized and optimized $Q_1$ for sparse and dense DTDs on a document of size 50MB. This yields several interesting results. Note that the graph of query evaluation time on sparse DTD in Figure 3.20 is flat, since the minimal SC annotation for each query node in $Q_1$ is (NC) or (LC). Thus, most of the security check is done locally. Also, query evaluation time decreases as the percentage of nodes with an assigned SecurityLevel attribute increases, because the recursive check needs to traverse fewer nodes to find the closest ancestors with an assigned security level.

We also tested speeding up query evaluation with optimized SC annotation versus unoptimized SC annotation against varying query size. A linear scaleup was observed: for example, for a query with 40 nodes the speedup was 6.11; At
Figure 3.18: Optimized SC annotation on query

80 nodes, we observed a speedup of 10.

3.6.3 Optimization Overhead

We tested how the overhead of optimization varies as query size increases. We ran the queries whose number of nodes varies from 10 to 30. Figure 3.21 shows optimization times for queries of various sizes over sparse and dense DTDs. Note that the optimization times on sparse DTDs are always smaller than on dense DTDs because there are fewer overriding nodes to check. Also note that the optimization time is very small: even for a query with 30 nodes it took only 0.297 seconds to optimize for the worst case. In all cases, we found the overhead to be a negligible fraction (e.g., 0.01%-1.1%) of the query evaluation time.

The placement of overriding nodes in a DTD can make a difference to optimization time. To measure this, we first tested the XMark, Biomedical and Congress datasets for overriding nodes crowding on very few DTD paths versus those nodes scattered among multiple DTD paths. Second, we tested on the same datasets for most of overriding nodes being leave nodes versus non-leaf nodes in the DTD. First, the difference in optimization time between “scattered” versus “lined up” nodes was substantial: for example, 0.02324 seconds versus 0.0141 seconds (about 40% less) for a DTD with 15 overriding nodes.
The reason is that when the nodes line up, the reasoning for one overriding node can subsume that required for the others, thus reducing the overall work. Second, we found that overriding DTD leaves contribute very little to the SC annotation optimization overhead. This is because the optimization overhead is decided by the number of intermediate overriding nodes that are reasoned during the optimization, in which the overriding leaves never get involved. For brevity, we have not shown these graphs.

Next, we varied the number of disjunction nodes in the disjunction DTD `bills.dtd` by randomly changing a few disjunction nodes in the query context of Q₃, Q₄ into sequence nodes. Figure 3.22 shows that the optimization overhead does not proportionally increase with the number of disjunction nodes, since the time complexity of optimization is dependent on the number of overriding nodes in the query context, not on the number of disjunction nodes.

### 3.7 Summary

In this chapter, we have considered the problem of secure evaluation of XML tree pattern queries, for the simple, but useful, multi-level access control model. We focus on an efficient algorithm that determines an optimal set of security-
check annotations with tree-pattern query nodes, by analyzing the subtle interactions between inheritance of security levels and the paths in the DTD graph. We experimentally validated our algorithm, and demonstrated the performance benefits to optimizing the secure evaluation of tree pattern queries.
Figure 3.22: Query Optimization Overhead with Regard to Number of Disjunction Nodes
Chapter 4

Efficient and Secure Query Evaluation over Encrypted XML Databases

In Chapter 3, we discussed how to securely and efficiently answer the queries over access-controlled XML databases. In this chapter, we move to encrypted XML database. In particular, we consider an encrypted XML database in the "database-as-service" model [50]. The idea of DAS model is that, for customers of limited budget (e.g., a small-sized enterprise) who have large volume of data, the database management is not affordable. This concern also applies to the customers who initially have small amount of data but eventually get data escalated to large volume. The long-term maintenance of data is expensive to them. Thus they use third parties that provide data management services to host, process and maintain their business and/or personal data. Although this option offers the possibility of reliable storage of large volumes of data, efficient query processing, and savings in database administration and maintenance costs for the data owner, it raises serious questions about the security of the data managed by the service provider. Given that no customer is willing to implicitly trust their data to another party, we need a mechanism to protect the privacy of sensitive data and to prevent security breaches. One solution to this problem is that the customer encrypts the sensitive parts of their data, sends the (partially) encrypted database to the service provider, and keeps the decryption key to himself/herself. We henceforth refer to the customer as the client or data owner, and the service provider as the server.

There are two kinds of information that the client may consider as sensitive: (i) the individual node with its content (here the content of the node consists of the tag, the sub-elements and the data values of leaf descendants) and (ii) the association between data values. For example, the owner of a health-care XML database may wish to protect every patient element. Alternatively, he or she may wish to protect the value associations between the patient's SSN and disease. To capture the client's security requirements, we propose the notion of security constraints (SCs) that support both types of security requirements identified above.

To enforce the security constraints specified by the data owner, we need a way to decide which elements in the database need to be encrypted. Encryption may be done at different granularities. For example, encrypting the entire docu-
ment, while guaranteeing that all security constraints are enforced, quashes any hopes of query optimization: for every query, the server needs to ship the whole encrypted database to the owner, who will decrypt and process the query, at a considerable burden. When the granularity is too fine, for example, only leaf elements are encrypted, as we will show in Section 4.3.1, an attacker can infer sensitive information, based on his background knowledge of the domain values and their occurrence frequencies. This scenario might violate one or more security constraints. Thus, one of the challenges is to find a “secure” and “optimal” encryption scheme (defined in Sections 4.2.4 and 4.3.2) that guarantees that security constraints are enforced while permitting efficient query processing.

After the data is encrypted and sent to the server, in order to enable efficient query processing, some metadata needs to be maintained on the server side. We want to ensure that the hosted database is “secure” in the presence of the metadata, i.e., that there are a large number of candidate databases, including the true hosted database, such that: (i) none of the candidate databases other than the true one contain any of the sensitive information being protected; (ii) by looking at the encrypted hosted data and the metadata, the attacker cannot distinguish between any pair of candidate databases.

Finally, we would like to make sure that even after observing a series of queries from the client and responses by itself, the server’s knowledge about the sensitive information in the database that is specified by the security constraints is always below a given security threshold.

The basic architecture and data/control flow of our system are shown in Figure 4.1. The client encrypts its database \(D\) to \(C\) by using some encryption function \(\eta\) and some private key known only to itself. The encryption may be partial (i.e., only a subset of the database is encrypted). The client also creates metadata \(M\) on \(D\) for the server’s use. Then he or she sends the encrypted database \(C\) together with the metadata \(M\) to the server. When a query \(Q\) needs to be evaluated on the database \(D\), the client translates \(Q\) into \(Q_s\), and sends \(Q_s\) to the server. The server answers \(Q_s\) by using the metadata \(M\). The answer to \(Q_s\), i.e., \(Q_s(C, M)\), consisting of a set of encrypted (and possibly some unencrypted) blocks, is sent to the client. The client decrypts the blocks as \(\delta(Q_s(C, M))\), using the decryption function \(\delta\), and does post-processing on the decrypted results using \(Q\), such that \(Q(\delta(Q_s(C, M))) = Q(D)\). We assume the client has some knowledge in XML and his queries are simple XPath queries. We assume the decryption and post-processing work can be done in memory. We assume the queries are non-aggregate queries.

**Contributions:** we make the following contributions in this chapter:

- We define the security properties for a hosted XML database system based on an attack model we define, consisting of size-based attack and frequency-based attack (Section 4.2).

- We propose a novel notion of security constraints (SCs) for the client to specify what information needs to be protected from the server (Section 4.2.2). This feature supports not only the protection of elements (both structure and contents) but also associations between data values
in the database. We propose a mechanism to construct a secure encryption scheme to enforce the security constraints (Section 4.3.1) and show that finding an optimal, secure encryption scheme for a given set of security constraints is NP-hard (Section 4.3.2).

- We propose a specific scheme for the metadata to be maintained by the server, to support efficient query processing. This scheme consists of two key components: (i) the structural metadata, called the discontinuous structural interval index (DSI), that pertains to the structure of the database; and (ii) a B-tree index as the metadata on the values that is based on transforming the unencrypted values by order-preserving encryption with splitting, scaling and noise techniques, so that the distribution of unencrypted data is different from that of the encrypted data. We show that even in the presence of metadata, the hosted database system is secure (Section 4.4).

- We show how queries can be evaluated efficiently and prove that query answering is secure. That is even when the attacker observes a series of client queries and server responses, his belief (probability) in whether the sensitive information captured by the given security constraints exists in the database, is always below a given security threshold (Section 4.5).

- We look into details of the overhead work on client side (Section 4.6). Specifically, we discuss the key management on the client side (Section 4.6.1) and show that although the client has a few activities to carry out, compared with the reduced cost in query processing and data maintenance, he/she is still benefited (Section 4.6.2).

- We complement our analytical results with a battery of experiments on both real and synthetic data sets that measure the effect of encryption scheme granularity and query size and shape on query processing efficiency, as well as the client post-processing cost, demonstrating the effectiveness of our approach (Section 4.7).

We will sometimes refer to unencrypted data values as plaintext and encrypted values as ciphertext. Related work of this chapter is discussed in Section
4.1. We summarize this chapter in Section 4.8. This work has been published on the International Conference on Very Large Database (VLDB) 2006 [89].

4.1 Related Work

Hacigumus et al. [50] were one of the earliest to study the DAS model. Their focus is the evaluation of SQL queries over hosted relational databases. They use conventional encryption techniques and additionally assign a bucket ID to each attribute value to facilitate efficient evaluation. The client needs to decrypt the results sent by the server and then executes a compensation query to extract the final answer. We share most of the goals of [50]. Agrawal et al. [8] have proposed a method of designing order-preserving encryption functions such that the distribution of the encrypted values follows an arbitrarily chosen target distribution. Neither of the above works consider user-specified security constraints. We focus on the problem of transforming the input data distribution such that by looking at both source and target distributions, the attacker cannot easily map the plaintext data to the ciphertext data, in a provable sense. This is a valuable technique since on the one hand, using order-preserving encryption, range queries (as well as aggregate queries involving MIN, MAX, COUNT) can be evaluated efficiently without decryption. At the same time, an attacker can be fooled by the target distribution. The notion of encryption scheme granularity is not an issue in [8]. Finally, unlike the above works, we consider a more conservative model where the attacker may know both the domain values and their exact occurrence frequencies. All of our security theorems are proven with regard to this background knowledge.

Protecting data privacy in relational databases has been addressed in various works (e.g., see [39], [51]). Kantarcioglu et al. [53] studied security issues in querying an encrypted database from a cryptographic viewpoint and showed that any secure database system that meets the strict cryptographic definitions of security will require a linear database scan for answering queries, and thus will be inefficient. Damiani et al. [36] proposed an indexing scheme and an efficient query evaluation scheme for encrypted relational data, and quantitatively measured the resulting inference exposure. They showed that the indexes proposed in [50] can open a door to interference and linking attacks.

Brinkman et al. [23] proposed a method of applying queries on encrypted XML documents. Their approach stores a relational table containing structural information of the XML database at the server, thus compromising the privacy of structural information. Also, they do not support flexible levels of encryption granularity. Bertino et al. [20] proposed differential encryption schemes for XML documents using multiple keys. Miklau et al. [71] proposed a framework for enforcing access control policies on published XML documents using cryptography. The security guarantee of this framework is precisely characterized by Abadi et al. in [1]. In another paper, Miklau et al. [72] studied the problem of query-view security and showed how to decide security based on a novel information-theoretic standard they proposed for analyzing the information dis-
closure of published views.

Song et al. [85] proposed a method of supporting search on plaintext files without loss of data confidentiality. Private information retrieval (PIR) (e.g., see [29]) is a related field of research. The pure information-theoretic notion of privacy requires that even a computationally unbounded server cannot distinguish between queries intended for different targets. It is not practical for database applications since it typically necessitates multiple copies and huge communication overheads. Computational privacy requires that polynomially bounded servers cannot distinguish between such queries. Asonov et al. [11] construct a PIR scheme with $O(1)$ online computation and communication, periodical off-line pre-processing, and zero off-line communication. Aggarwal et al. [6] proposed partitioning the database between two servers so that associations between data items can be kept private from each of the servers (in an intuitive but not cryptographic sense), but can be inferred by the client. A basic assumption is that the servers cannot communicate with each other. One-server schemes with subpolynomial communications were proposed in [55]; however, they impose expensive computational overhead. Liu and Candan [63] propose a computationally PIR one-server scheme that achieves both content and access privacy with moderate computation and communication cost. However, this scheme comes at the price of access redundancy. The client has to employ so-called node swapping (i.e., swapping two nodes in the tree to change their positions), which can be expensive.

There has been significant research on access control, privacy-preserving data mining, secure XML publishing, and trusted computation. While these research shares our concern for security and privacy, their assumptions, goals, and other concerns are considerably different. We refer the reader to [6] for related work in these topics.

### 4.2 Security Issues

In this section, we define the concepts of security constraints, attack model, and present several security definitions.

#### 4.2.1 Encryption Scheme

Given an XML database $D$, we define an encryption scheme $S$ as consisting of (1) an encryption function $\eta$, and (2) an identification of those elements that are to be encrypted by $\eta$. Every element is encrypted as a block. We assume every encrypted block is assigned a unique ID. Figure 4.2 shows an example of such a scheme. The encryption scheme is illustrated by highlighting encrypted blocks with boxes. We have a choice in the granularity of the encryption scheme, e.g., encrypt the whole document, encrypt (some of) the leaves, or encrypt something in between. The choice should be driven by what it is that we wish to protect, and by the desire to minimize the overhead imposed upon the client for extracting query answers.
4.2.2 Security Constraints

We propose security constraints (SCs) as a means for the data owner to specify the information, i.e., the set of basic queries, that he or she intends to protect from an attacker. To be precise, a security constraint is an expression of the form \( p \) or of the form \( p : (q_1, q_2) \), where \( p \), \( q_1 \), and \( q_2 \) are XPath expressions. The security constraint \( p \), called the node-type constraint, specifies that for every node \( x \) in the document \( D \) that \( p \) binds to, the element subtree rooted at \( x \), consisting of its tag, its content, and structure, is considered as classified. Note that the attacker may have some prior knowledge of existence of some tags in the subtree rooted at \( x \). But he does not know the exact position of these tags in the database as well as their ancestors and descendants. For this case, we still consider \( x \) is securely protected if from a set of candidates, the attacker cannot explicitly decide which is the original subtree rooted at \( x \).

The security constraint \( p : (q_1, q_2) \), called the association-type constraint, says that for every node \( x \) that \( p \) binds to and for every pair of nodes \( y_1, y_2 \) containing data values \( v_1, v_2 \), that \( q_1, q_2 \) bind to in the context of \( x \), the association between \( v_1 \) and \( v_2 \) is considered to be classified.

Example 4.2.1 [Security Constraints]

For the database in Figure 4.2, the client intends to protect the following information: (1) the insurance information of each patient, (2) which SSN number matches with which patient’s name, (3) which patient has which diseases, and (4) which doctor treats which diseases. The security constraints are specified as follows:

\( SC_1: //\text{Insurance} \), to protect \( \text{Insurance} \) elements,
\( SC_2: //\text{patient}://\text{name} , //\text{SSN} \), to protect the association between patient’s name and SSN number,
\( SC_3: //\text{patient}://\text{name} , //\text{disease} \), to protect the association between patient’s name and disease,
\( SC_4: //\text{treat}://\text{disease} , //\text{doctor} \), to protect the association between the
Each security constraint can be seen as capturing a set of queries. A node
type security constraint \( p \) captures queries of the form \( p \), as well as queries
\( p/a, p//a, p[a\text{[following-sibling :: }b]\text{]} \) etc., where \( a, b \) are element tags. The
idea is that whenever \( p \) binds to an element \( x \) in the database \( D \), whether
these queries have a non-empty answer in the database \( D \), as well as the
answers, should be protected. For example, in Figure 4.2, the security con­
straint //Insurance captures the queries //Insurance, Insurance/Policy#, Insurance//*/@coverage, //Insurance//Policy#, etc..

An association security constraint of the form \( p : (q_1, q_2) \) captures
the queries \( p[q_1 = v_1][q_2 = v_2] \) for all values \( v_1, v_2 \) such that the XPath expression above has
a non-empty answer in the database \( D \). For example, the association security
constraint //patient : (pname, //disease) captures the queries //patient
[pname = Betty][//disease = diarrhea] and //patient[pname = Matt]
[//disease = leukemia]. Whether these queries have a non-empty answer must
be protected from the attacker.

For a security constraint \( \sigma \) and a query \( A \) captured by \( \sigma \), we denote by
\( D \models A \) that \( A \) has a non-empty answer in the database \( D \). Thus, for each
query \( A \) captured by each SC, we need to protect the fact that \( D \models A \). We
implement protection by encryption. Intuitively, enforcing the security constraints
requires encrypting certain elements in the database. For example, the enforce­
ment of the security constraint //Insurance can be done by encrypting all
Insurance elements. The security constraint //patient : (pname, //disease)
can be enforced by encrypting either pname elements or disease elements. Fig­
ure 4.2 shows an example of an encrypted database. The encryption scheme is
illustrated by highlighting encrypted elements with boxes.

4.2.3 Prior Knowledge and Attack Model

We assume that the server/attacker may be curious and try to learn about the
data it is hosting, but that it never modifies the data and always answers queries
correctly. We consider the following two kinds of attacks by the server/attacker:
Frequency-based Attack: We assume that the attacker possesses prior knowl­
dge about the domain values and the exact occurrence frequencies for attributes
and leaf elements in the hosted database. By comparing the distribution of original
plaintext and ciphertext values, the attacker may match ciphertext values to
plaintext values having the same frequency, and thus try to crack the database.
This type of attack is called a frequency-based attack. The data distribution
knowledge can be collected from either the encrypted database or the meta­
data. Thus, frequency-based attack may be based on the encrypted database
or the metadata.
Size-based Attack: The attacker may also possess the common knowledge
that the length of plaintext decides that of ciphertext. Two plaintexts of the
same length will result in corresponding ciphertexts of the same length. Thus
if the attacker has a set of candidate plaintext databases for the hosted data,
he will try to eliminate those candidate databases whose encrypted result is of different length from that of the encrypted hosted data, and consequently increase the probability of the remaining candidates to be the hosted data. We call this attack size-based attack.

### 4.2.4 Security Definitions

We first define the indistinguishability of databases with regard to the above two possible attacks. Let \(|D|\) denote the size of the database being considered, whether encrypted or not.

**Definition 4.2.1 [Indistinguishability of Databases]** Let \(S\) be an encryption scheme. Let \(D, D'\) be two plaintext XML databases. We say \(D\) and \(D'\) are indistinguishable, written \(D \sim D'\), if:

1. \(|S(D)| = |S(D')|\)
2. For each attribute, \(D\) and \(D'\) have the same domain of values, and for each domain value \(v\), the corresponding encrypted value in \(S(D)\) and \(S(D')\) has the same occurrence frequency.

This definition ensures that the attacker cannot distinguish \(D\) and \(D'\) by either size-based attack (Condition (1)) or frequency-based attack (Condition (2)). Thus, if there is a set of candidate databases including the true hosted database and they are all pairwise indistinguishable, we can say that those candidate databases have equal probability of being the true hosted database.

Although the candidate databases may be equally likely, when the attacker has some prior belief probability that each candidate database is the true hosted database, the attacker may be able to infer something when looking at the encrypted hosted database. We then use perfect security as the security requirement for this case. Perfect security [82] is a term used in cryptography. An encryption scheme has perfect security provided the probability of obtaining the corresponding plaintext message \(M\) for a given encrypted (i.e., ciphertext) \(C\), is the same as obtaining the plaintext without any ciphertext to observe, i.e., the attacker is unable to obtain any additional information from the ciphertext.

We adapt this definition to databases.

**Definition 4.2.2 [Perfect Security]** Let \(C\) be an encrypted XML database and let \(D_1, D_2, \ldots, D_n\) be a set of plaintext databases, each with a priori probability \(\text{Prob}(D = D_i)\) of being the true hosted database. Let \(\text{Prob}(D = D_i | C = C)\) be the a posteriori probability of \(D_i\) being the true hosted database given the encrypted hosted database \(C\). We say \(C\) is perfectly secure if \(\text{Prob}(D = D_i | C = C) = \text{Prob}(D = D_i)\), where \(D\) (\(C\)) denotes the plaintext database (encrypted database).

That is, upon seeing the encrypted database, the attacker cannot change his/her a priori probability of any \(D_i\) being the true hosted database.

Based on the definitions of indistinguishability and perfect security, we define the following secure encryption scheme:
Definition 4.2.3 [Secure Encryption Scheme] Let $D$ be any XML database, $\Sigma$ be a set of security constraints on $D$, and $S$ be an encryption scheme. Let $S(D)$ be the ciphertext database of $D$ encrypted according to $S$. We say $S$ is secure on $D$ if for any security constraint $\sigma \in \Sigma$, and for every query $A$ captured by $\sigma$, there exists a large number of databases $D'$ such that: (1) $D' \sim D$, (2) $D \models A$, but $D' \not\models A$, and (3) $S(D)$ is perfectly secure.

This definition requires that for a large set of candidate databases that do not contain any sensitive information (Condition (2)), by looking at the encrypted hosted database alone, the attacker can distinguish them neither from each other nor from the original hosted data (Condition (1)), nor can he/she increase the probability that any of them is the true hosted data (Condition (3)). This definition formalizes what it means for an encryption scheme to enforce a given set of security constraints on a database. We will show in Section 4.3 that our proposed encryption scheme is secure. The term "large" is intentionally left undefined in Definitions 4.2.3. When proving security properties, we will actually show that typically "large" means exponential in the size of the domain or the schema.

To facilitate efficient query processing, we need to place some metadata on the server. However, by looking at both the encrypted hosted data and the metadata on the server, the attacker should not be able to infer any sensitive information of the original database. The following definition addresses this requirement. Before we give the definition, we first define metadata function. Let $\mathcal{D}$ be the domain of the plaintext databases. Let $\mathcal{C} = \{S(D) \mid D \in \mathcal{D}\}$ be the domain of the encrypted databases, where $S$ is the encryption scheme. We define a metadata function $\mu : \mathcal{D} \times \mathcal{C} \rightarrow \Omega$, where $\Omega$ is an abstract domain that possibly can be of any data type. Intuitively, the metadata is constructed from the plaintext database $D$ and the encrypted version of $D$ by applying the function $\mu$ on them. We require that $\mu$ satisfies: for any database $D \in \mathcal{D}$ and any query $Q$ on $D$, $Q(D) = Q(\delta(Q_s(S(D), \mu(D, S(D))))))$, where $Q_s$ is the translated version of $Q$ that is applied on both $S(D)$ and $\mu(D, S(D))$, and $\delta$ is the decryption function. Given a query $Q$, for any two databases $D$ and $D'$, we say $Q(S(D), \mu(D, S(D))) = Q(S(D'), \mu(D', S(D'))))$ if they return the same set of IDs of encryption blocks. Given an XML plaintext database $D$, we call $DS(D, S(D), \mu(D, S(D)))$ a hosted XML database system. We have:

Definition 4.2.4 [Secure Database System] Given a hosted XML database system $DS(D, S(D), \mu(D, S(D)))$, where $D$ is the plaintext database, $S(D)$ is the encrypted version of $D$, and $\mu(D, S(D))$ is the metadata on $D$, let $\Sigma$ be a set of security constraints on $D$. Let $K$ be an user-specified security threshold. We say the hosted XML database system $DS(D, S(D), \mu(D, S(D)))$ is secure if there exist at least $K$ databases $\{D'_1, D'_2, \ldots, D'_K\}$ such that: (1) $\mu(D, S(D)) = \mu(D'_i, S(D'_i))$ (1 $\leq i \leq K$), (2) $D \sim D'_i$ (1 $\leq i \leq K$), (3) for every query $A$ captured by any security constraint in $\Sigma$, $D \models A$, but $D'_i \not\models A$ (1 $\leq i \leq K$), and (4) for every query $A$ captured by any security constraint in $\Sigma$, let $A_s$ be the translated query of $A$, $A_s(S(D), \mu(D, S(D))) = A_s(S(D'_i), \mu(D'_i, S(D'_i)))$ (1 $\leq i \leq K$).
This definition requires that there exist at least $K$ candidate databases that do not contain any sensitive information (Condition (3)). Furthermore, for any query that is captured by any security constraint, these $K$ candidate databases return the same query results as the true database (Condition (4)). However, by looking at the encrypted database as well as the metadata, the attacker cannot distinguish them from each other or from the original hosted data (Conditions (1) and (2)). We will show in Section 4.4 that by using our proposed metadata (consisting of structural and value indices), the resulting database system is secure.

Note that if the attacker only applies the size-based attack and the frequency-based attack, every candidate database is equally likely to be the true hosted database. However, by observing the metadata, it is possible that the attacker can further infer that some of the candidate databases are fake, so that his/her a posteriori probabilities of these databases to be the true database changes to zero, which does not equal to their a priori probabilities. Therefore, perfect secrecy (as in Definition 4.2.3) may no longer be satisfied in Definition 4.2.4. However, since by specifying the security threshold $K$ in Definition 4.2.4, we guarantee that there still exists a large set of candidate databases from which the attacker cannot tell the fake candidates from the true one, we consider it is reasonably secure.

Finally, we also consider an attacker’s possible inference while observing query answering. To make this problem precise we consider the following predicates: let $A$ be any query captured by a security constraint. Let $B$ denote an encryption block (i.e., an element that is encrypted). Then we use $B(A)$ to denote that block $B$ satisfies query $A$. Note that $A$ may be either structure-based or value-association-based. The question is whether the server’s belief probabilities in these assertions $B(A)$ is always under a small security threshold after observing a series of queries and answers. We assume the server has no prior knowledge about the query workload distribution. We address this concern in the form of the following definition of secure query answering.

**Definition 4.2.5 [Secure Query Answering]** Let $\Sigma$ be a given set of security constraints, $D$ be a database, $S(D)$ be the encrypted database obtained from $D$ using an encryption scheme $S$, and $\mu(D, S(D))$ be the metadata on $D$. Let $A$ be any query captured by any security constraint in $\Sigma$ and let $\text{Bel}_q(B(A))$ denote the attacker’s belief probability that the proposition $B(A)$ holds for any encryption block $B$, after observing $q$ queries and responses. Let $\gamma$ be an user-specified security threshold. Then we say that the query answering by the server is secure if $\forall q \geq 0$, $\text{Bel}_q(B(A)) < \gamma$ holds after he/she observes $q$ queries and responses.

We will show in Section 4.5 that our query processing procedure is not only efficient but also secure.
4.3 Encryption Scheme

Given an XML database $D$, an encryption scheme consists of (1) an encryption function $\eta$, and (2) an identification of those elements that are to be encrypted by $\eta$. An XML document tree can be encrypted at any level of granularity, e.g., the whole document, chosen elements at any depth, or just the content (text values) of chosen elements. In choosing a level of granularity for encryption, there is a trade-off between the data security and efficiency of query evaluation. Too coarse-grained encryption schemes, e.g., encrypting the whole database into one or very few blocks, can quash hopes of query optimization, while too fine-grained encryption schemes may allow an attacker to learn about the data, thus violating security requirements. In this section, we first show how to construct a secure encryption scheme in Section 4.3.1. Then in Section 4.3.2, we define the optimal, secure encryption scheme and discuss the complexity of finding an optimal, secure encryption scheme.

4.3.1 Secure Encryption Scheme

Although the plaintext data is hidden after encryption, careless design of the encryption scheme may still allow an attacker to infer the plaintext data by frequency-based attack. For example, assume the association between patient's age 40 and his disease leukemia in Figure 4.2 should be protected. If the client plainly encrypts each disease and age element individually, the encrypted value of leukemia will have the same number of occurrences as before encryption, which is also true for value 40. Then by matching the frequency of occurrences of plaintext and ciphertext values, even with both disease and name encrypted, the attacker can easily identify the plaintext values and infer the classified association. We assume only the distribution on the leaf nodes are known to the attacker.

From the above example, we can see that encrypting leaf nodes is not safe. To address the security issue, we use one-time-pad scheme (also called the Vernam cipher) for encryption. One-time-pad encryption requires that every encryption key can only be used once. We make use of this and choose the keys in the way such that for any two elements $e, e'$, the results of encrypting them with keys $k, k'$ are not identical. For example, for the two disease nodes in Figure 4.2 of value diarrhea, they will be encrypted into two distinct ciphertext values.

Let $S$ be an encryption scheme, $D$ be a database, and $\Sigma$ be a set of security constraints. Let $S(D)$ be the encrypted database according to $S$, such that: (i) for every node type security constraint $p \in \Sigma$, $p$ is encrypted by one-time-pad scheme, and (ii) for every association type security constraint $p : (q_1, q_2) \in \Sigma$, the nodes that bind either to $p/q_1$ or to $p/q_2$ are encrypted by one-time-pad scheme, and (iii) every leaf element that is encrypted is encrypted by one-time-pad scheme with unique output. We present the following theorem.

---

$^1$In this chapter we assume that the data values in the XML documents are only attached to the leaf nodes and do not consider mixed content.
Theorem 4.3.1 (Secure Encryption Scheme): Let $S, D, \Sigma, S(D)$ be as above. Then $S$ is a secure encryption scheme (Definition 4.2.3).

To prove the correctness of Theorem 4.3.1, we first present Lemma 4.3.1 and Lemma 4.3.2.

Lemma 4.3.1 (Exponentiality of $\frac{(\sum_{i=1}^{n}k_i)!}{k_1!k_2!...k_n!}$): For a set of positive integers $k_1, k_2, \ldots, k_n$ ($n \geq 2$), $\frac{(\sum_{i=1}^{n}k_i)!}{k_1!k_2!...k_n!}$ is exponential in $\sum_{i=2}^{n} k_i$.

Proof:
Let $F(n) = \frac{(\sum_{i=1}^{n}k_i)!}{k_1!k_2!...k_n!}$. We have:

$$F(n) = \frac{(\sum_{i=1}^{n-1}k_i)!+1}{(\sum_{i=1}^{n-1}k_i)!+2} \times \frac{\sum_{i=1}^{n-1}k_i+1}{\sum_{i=1}^{n-1}k_i}$$

Since $t \leq k_n$, $\frac{(\sum_{i=1}^{n-1}k_i)!+t}{\sum_{i=1}^{n-1}k_i}$, we have $F(n) \geq (\frac{\sum_{i=1}^{n-1}k_i}{\sum_{i=1}^{n-1}k_i})^n F(n-1)$. By "unrolling" $F(n)$, we have:

$$F(n) \geq (\frac{\sum_{i=1}^{n-1}k_i}{\sum_{i=1}^{n-1}k_i})^n F(n-2) \times (\sum_{i=1}^{n-1}k_i) F(n-1)$$

Let $f_{min} = \min\{\frac{\sum_{i=1}^{n-1}k_i}{\sum_{i=1}^{n-1}k_i}, \frac{\sum_{i=1}^{n-1}k_i}{\sum_{i=1}^{n-1}k_i}, \ldots, \frac{k_1+k_2}{k_2}\}$. Since $F(1) = x_1!/x_1! = 1$, we have: $F(n) \geq f_{min} \times \frac{k_1+k_2}{k_2}, \ldots, \frac{k_1+k_2}{k_2}$, which is exponential in $k_2 + k_3 + \ldots + k_n$.

In stating (and proving) Lemma 4.3.1, we order the $k$'s as $k_1, k_2, \ldots, k_n$. This ordering has the effect of eliminating $k_1$ from the sum, i.e., $F(n)$ is exponential in $\sum_{i=2}^{n} k_i$. But we could have ordered the $k$'s differently, e.g., $k_2, k_1, k_3, \ldots, k_n$. Then $k_2$ would be eliminated instead, and so on. Thus, $F(n)$ is exponential in $\max_{1 \leq i \leq n} \sum_{1 \leq i \leq n, i \neq k_i}$.

And we have:

Lemma 4.3.2 (Probability): Let $D$ be a set of plaintext databases that are pairwise indistinguishable (Definition 4.2.1). Let $C$ be a set of encrypted databases, where each is generated by applying the encryption scheme $S$ in Theorem 4.3.1. Let $\text{Prob}(C = C | D = D)$ be the probability that the encrypted database is $C$ given the true hosted database is $D$. Then $\text{Prob}(C = C | D = D) = 1/|C|$.

Proof: Given the true hosted database $D$, since every plaintext database in $D$ has the same size (following definition of "indistinguishable"), the encrypted databases in $C$ that are encrypted by one-time-pad scheme must have the same size too. Furthermore, since all ciphertext databases have the same frequency distribution, they are pairwise indistinguishable according to both size-based attack and frequency-based attack. Additionally, due to one-time-pad scheme, each database in $C$ is chosen equiprobably by the attacker, even provided the attacker has unbounded computational power. Therefore, $\text{Prob}(C = C | D = D) = 1/|C|$.

Based on Lemma 4.3.1 and Lemma 4.3.2, we prove Theorem 4.3.1.

Proof of Theorem 4.3.1:
Assume that a leaf element/attribute has $n$ distinct plaintext values in the original database $D$, each with an occurrence frequency $k_i$. By observing the encrypted database, the attacker will have correspondingly $m$ distinct ciphertext values, each with occurrence frequency 1, where $\sum_{i=1}^{n} k_i = m$. But simply matching the frequency of the ciphertext to that of plaintext does not help the attacker to crack the plaintext value. Each plaintext value can be mapped to a set of ciphertext values. The only possibility is to try out all such mappings.

The number of mappings corresponds to the number of ways of partitioning $m$ into $n$ non-zero parts. This number is $N = \frac{(m)}{k_1} \times (m-k_1) \times \cdots \times (m-k_1-\cdots-k_{n-1}) = \frac{(m)!}{(k_1)!(k_2)!\cdots(k_n)!}$. This is the number of candidate databases the attacker has to consider, where we have taken only one leaf element/attribute into account. For the $\frac{(m)!}{(k_1)!(k_2)!\cdots(k_n)!}$ databases, the following holds true:

\[ \text{Exponentially "large".} \quad \frac{(m)!}{(k_1)!(k_2)!\cdots(k_n)!} \text{ is exponential in } \sum_{i=1}^{n} k_i \text{ (as proven in Lemma 4.3.1), and thus satisfies the "large" requirement in Definition 4.2.3. For example, for } k_1=3, k_2=4, k_3=5, N=(3+4+5)! / (3!4!5!) = 27720. \]

Condition (1) of Definition 4.2.3: For each database $D'$, $S(D')$ has the same domain and the same frequency distribution of encrypted values as $S(D)$. Furthermore, each $S(D')$ has the same structure as $S(D)$. Thus, they are of the same size. Therefore, $D' \sim D$.

Condition (2) of Definition 4.2.3: Out of those $N$ candidates, there is only one $D$ whose mapping of plaintext values and ciphertext values is correct. Thus $D = A$, where $A$ is a given security constraint. However, for the other candidate databases $D' \neq D$, $D' \not\sim A$.

Condition (3) of Definition 4.2.3: Finally, for those $N$ candidates $D$ and the given encrypted hosted database $C$, since candidate every database in $D$ are pairwise indistinguishable by either the size-based attack or the frequency-based attack (as proven by Condition (1) above), we can prove it satisfies the definition of perfect secrecy (Definition 4.2.2), i.e., $\text{Prob}(D = D | C = C) = \text{Prob}(D = D)$. Based on Lemma 4.3.2, we have:

\[ \text{Prob}(D = D | C = C) = \frac{\text{Prob}(C = C | D = D) \times \text{Prob}(D = D) \times \text{Prob}(C = C)}{\text{Prob}(D = D) \times 1/|C|} = \frac{1/|C| \times \text{Prob}(D = D) \times \text{Prob}(C = C)}{\text{Prob}(D = D) \times 1/|C|} \]

And

\[ \text{Prob}(C = C) = \sum_{D_i} (\text{Prob}(D = D_i) \times \text{Prob}(C = C | D = D_i)) = \sum_{D_i} \text{Prob}(D = D_i) \times 1/|C|. \]

Since $\sum_{D_i} \text{Prob}(D = D_i) = 1$, $\text{Prob}(C = C) = \text{Prob}(C | D = D) = 1/|C|$ (**) By plugging (** into (*), we have

\[ \text{Prob}(D = D | C = C) = 1/|C| \times \text{Prob}(D = D) \times (1/|C|) = \text{Prob}(D = D). \]

Thus the encrypted database $C$ is perfectly secure.  

In summary, our encryption scheme is secure. Figure 4.2 shows an example of a secure encryption scheme that enforces the security constraints defined in
Example 4.2.1. Since one-time-pad encryption requires a key as long as the total amount of data encrypted, and is not practical, we use block cipher in cipher-block chain (CBC) mode with initial vector IV [46] for the implementation. A nice property of this mode is that every encrypted message is unique. However, it does not satisfy perfect secrecy, especially for the attacker who has unbounded computational power.

4.3.2 Optimal Secure Encryption Scheme

For a given set of security constraints, there may be multiple secure encryption schemes. Using the database in Figure 4.2 as an example, to protect the disease data, one of the secure encryption schemes is to encrypt the whole document as a block. Some other possible encryption schemes are: encrypt the individual treat elements, or encrypt individual disease node. Different encryption schemes have different query-processing performances with regard to the cost of transmission, decryption and query post-processing.

To address the efficiency issue of query processing, we define the optimal (and secure) encryption scheme. Notice that each encryption scheme can be equivalently viewed as a set of encryption blocks. Thus, we abuse the notation and write \( b \in S \), for an encryption block \( b \) in an encryption scheme \( S \). The size of an encryption block \( b \) is the number of nodes in it. The size of an encryption scheme \( S \) is the sum of sizes of its encryption blocks, i.e., \( |S| = \sum_{b \in S} |b| \).

**Definition 4.3.1 [Optimal Secure Encryption Scheme]** Let \( \Sigma \) be a set of user-specified security constraints on a database \( D \). Then an optimal secure encryption scheme \( S \) on \( D \) that is of the smallest size, i.e., for all \( S' \) that are secure on \( D \), \( |S| \leq |S'| \).

The optimal secure encryption schemes are not unique. For example, to enforce the security constraints specified in Example 4.2.1 on the instance shown in Figure 4.2, besides encryption on two Insurance nodes individually, we can encrypt two name nodes and three disease nodes, or we can encrypt two SSN nodes and three disease nodes. Both encryption scenarios have the same size. We next address the complexity of finding an optimal secure encryption scheme.

**Theorem 4.3.2 (Complexity of Optimal Secure Encryption)**: Given an XML database \( D \) and a set of security constraints \( \Sigma \), finding an optimal secure encryption scheme for \( \Sigma \) on \( D \) is NP-hard.

**Proof**: We prove the NP-hard complexity result, the complexity of which is in the size of the security constraints. The proof is by reduction from VERTEX COVER, an NP-complete problem [32]. The problem is, given a graph \( G = (V, E) \), find a vertex cover \( C \subseteq V \). Given an instance of this problem, we create an XML tree \( D \) with a root element \( r \) with child element types corresponding to nodes in \( G \). For every edge \( \{x, y\} \in E \), there exists an association constraint \( r : \{x, y\} \) on \( D \). It can be seen that \( C \subseteq V \) is a minimum vertex cover of \( G \) if
and only if the encryption scheme that encrypts instances of each element type in \( C \) is an optimal encryption.

Given the NP-hardness result, we seek approximations. Indeed, vertex cover has several approximation algorithms (see Motwani [73]). In principle, we can adapt any of them to devise an algorithm for finding an encryption scheme whose cost is no worse than twice the optimal cost.

The best known deterministic approximation algorithm is by Ber-Yehuda and Even [73], and it achieves a ratio of \( 2 - O(\log \log n / \log n) \) times the size of the optimal solution. To leverage this algorithm, we show how to reduce the optimal encryption granularity problem to a vertex cover problem (i.e., reduction in the other direction). Given an XML database \( D \) and a set of security constraints \( E \), we construct the graph \( G \) such that for every node-type constraint \( p \) in \( \Sigma \), \( G \) contains a node corresponding to \( p \), and for every association-type constraint \( p : (q_1, q_2) \), \( G \) contains two nodes corresponding to \( q_1 \) and \( q_2 \), as well as the edge corresponding to the association between them. We call the resulting graph the constraint graph. It is easy to see a 1-1 correspondence between the covers of the constraint graph and the encryption schemes for \( D \) and \( \Sigma \). From this correspondence, it follows that any approximation algorithm for vertex cover can be applied for obtaining an encryption scheme that is no more than two times worse in the term of size than an optimal scheme.

### 4.4 Metadata on Server

To enable efficient query processing, we propose to add some metadata on the hosted data at the server. Our proposal consists of two parts: a structural index and a value index. In this section, we will present techniques for these two indices and show that when using a secure encryption scheme enforcing given security constraints, in the presence of the proposed indices, the database system as a whole is secure.

#### 4.4.1 Structural Index

Recall that the server stores encrypted blocks, corresponding to subtrees of the XML data, in addition to unencrypted data, as determined by the chosen encryption scheme (e.g., see Figure 4.2). We want the server to be able to efficiently locate the encryption blocks as well as unencrypted data nodes that satisfy the user query without the server learning any sensitive information.

We propose a novel index scheme, called a discontinuous structural interval (DSI) index,² as an effective way to index tree-structured data. The DSI index for a database tree \( D \) is obtained as follows. The root is assigned the interval \([0, 1]\). Inductively, children of an internal node \( p \) are assigned subintervals of \( p \)'s interval, but gaps are introduced between the intervals of (1) \( p \) and its first child, (2) adjacent children of \( p \), and (3) the last child of \( p \) and \( p \) itself. The gaps will make the intervals of adjacent nodes discontinuous. We differentiate

²As contrasted with the well-known continuous interval indexing scheme [9].
Function callInterval()

Input: Node p with interval $[\text{min}, \text{max}]$, weight $w_1$, $w_2$

Output: Interval of p's $i$th child $c_i$, $[\text{min}_i, \text{max}_i]$

Assume p has $N$ children.

d=$(\text{max} - \text{min})/(2 \times N + 1)$

\[ \text{min}_i= \text{min} + (2 \times i - 1) \times d - d \times w_1 \]

\[ \text{max}_i= \text{min} + 2 \times i \times d + w_2 \times d \]

Figure 4.3: Calculating DSI Index

the gap length by assigning two arbitrary weights $w_1$ and $w_2$ (known only to the client), where $w_1, w_2 \in (0, 0.5)$ is a real number. The weights are generated uniformly before assigning an interval to a node. Then we calculate the interval of every node by using the algorithm shown in Figure 4.3, assuming we know node p's interval $[\text{min}, \text{max}]$, and calculate its child intervals.

For any two adjacent nodes $p_i$ and $p_{i+1}$ with intervals $[\text{min}_i, \text{max}_i]$ and $[\text{min}_{i+1}, \text{max}_{i+1}]$, we have:

\[
\text{max}_{i+1} = \text{min}_i + (1 + w_{1}^2 + w_{2}^2)d,
\]

\[
\text{min}_{i+1} = \text{min}_i + (2 - w_{1}^2 + w_{2}^2)d.
\]

$d$ is defined in Figure 4.3).

The gap between these two adjacent nodes $\text{min}_{i+1} - \text{max}_i = (1 - w_{1}^2 - w_{2}^2)d > 0$.

In particular, the lower bound of the first child is larger than that of its parent, i.e., $\text{min}_i > \text{min}$, and the upper bound of the last child is less than that of the parent, i.e., $\text{max}_N < \text{max}$.

Proposition 4.4.1 immediately follows the definition of the DSI index. The terms “follows” and “following sibling” in Proposition 4.4.1 are of the same semantics of XPath axes “following” and “following-sibling” (the definitions of these axes can be found in Section 2.1.2).

Proposition 4.4.1 (DSI Index): Let $D$ be an XML database tree containing nodes $x, y$ with intervals $[x.\text{min}, x.\text{max}]$ and $[y.\text{min}, y.\text{max}]$. Then

1. $\text{descendant}(x, y) \leftrightarrow x.\text{min} > y.\text{min}, x.\text{max} < y.\text{max}$,
2. $\text{child}(x, y) \leftrightarrow \text{descendant}(x, y) \land \exists z : \text{desc}(x, z) \land \text{desc}(z, y)$,
3. $\text{follows}(x, y) \leftrightarrow x.\text{min} > y.\text{max}$, and
4. $\text{following-sibling}(x, y) \leftrightarrow \text{follows}(x, y) \land \text{child}(x, z) \land \text{child}(y, z)$.

The correctness of Proposition 4.4.1 is straightforward since our DSI index has similar principles as continuous index to indicate the positions of nodes in the database. Proposition 4.4.1 shows how DSI index can support efficient checks on the child, descendant, following and following-sibling XPath axes. The checks on ancestor, parent, preceding and preceding-siblings are symmetric to the above predicates. A notable exception is child. Rather than store level number for nodes, which would give away too much structural information, we compute child somewhat less efficiently by using $\text{child}(x, y) \leftrightarrow \text{desc}(x, y) \land \exists z : \text{desc}(x, z) \land \text{desc}(z, y)$. In general, the data owner has to decide whether this slight performance penalty is worth the added security it buys.
Updates
The DSI index is incrementally updatable due to the introduced gaps. Whenever an existing node is deleted, the intervals of all other existing nodes are left unchanged. When a new node is inserted, it is inserted into the gap by evenly splitting it into three subintervals. The middle subinterval is used as the interval for the new node and the other two are left as new gaps between the new node and its adjacent nodes. The intervals of all existing nodes are left unchanged. Note that it is possible that if there are many insertions, the intervals of new nodes may be out of precision. Hence, our scheme is suitable for the databases of infrequent updates.

The details are shown in Figure 4.4. If the new node is inserted as the first child of node \( x \), \( x \) will be treated as node \( p_1 \) and the current first child of \( x \) will be treated as node \( p_2 \). Similarly, if the new node is inserted as the last child of node \( x \), \( x \)'s current last node will be treated as \( p_1 \) and \( x \) will be treated as \( p_2 \).

4.4.2 Representing DSI index

DSI Index Table
We represent the DSI index using the DSI index table. It stores the mapping between the tags (in encrypted format, if the corresponding element was encrypted in the database) and their DSI index entries. We use the one-time-pad encryption scheme (also called the Vernam cipher[46]) for the tag encryption, because of its perfect security property. Note that the total size of keys do not equal to the size of encrypted data, but the size of distinct encrypted tags.

It should be noted that simply storing all of the intervals reveals too much information to the server (see Example 4.4.1 below). Thus we do the following: for those adjacent nodes that are of the same tag and are encrypted in the same block, we group (part of) their intervals into one by using the lower bound of the left-most node as the lower bound, and the upper bound of the right-most node as the upper bound. For example, the two adjacent nodes “Policy#” in Block 2 (Figure 4.2) will be represented by a single interval \([0.399, 0.433]\). Figure 4.6 (a) shows part of the DSI index table of the instance in Figure 4.2. To satisfy the security definition (Definition 4.2.4 and 4.2.5), we require that for a database that is encrypted into \( m \) blocks with each encryption block \( i \) containing \( n_i \) leaf nodes, the number of intervals \( k_i \) needs to satisfy: (1) \( \Pi_{i=1}^m (\frac{n_i}{k_i}-1) > K + 1 \),
and (2) \( \left( \frac{n-1}{k} \right) > 1/\gamma \), where \( K \) and \( \gamma \) are two given security thresholds in Definitions 4.2.4 and 4.2.5. The reason for this constraint will be shown in the proof of Theorems 4.4.1 and 4.5.1.

Note that if the same grouping technique is applied on a set of sibling nodes when a continuous indexing scheme (for example, the scheme described in [9]), the grouping may cause discontinuity of the index, and consequently the server may discover the existence of grouping, and further possibly the exact structure of the tree. Example 4.4.1 shows such information leakage on the continuous index.

**Example 4.4.1 [Insecurity of Continuous Interval Index]**

Figure 4.5 (a) shows a tree instance that is labeled with a set of continuous interval index. We assume that this tree is encrypted in one block. The corresponding index table after grouping is shown in Figure 4.5 (b). It can be seen from the index table that there is a gap between D's interval \([46, 259]\) and one of C's intervals \([262, 498]\), which is invalid against the continuousness property of the indexing scheme. Thus the attacker can infer that interval \([2, 499]\) of node B must at least contain two B nodes, one of interval \([2, 260]\) and the other of interval \([261, 499]\), i.e., the grouping on B is revealed to the attacker. |
Chapter 4. Query Evaluation over Encryption XML Databases

<table>
<thead>
<tr>
<th>Tag</th>
<th>DSI Index</th>
<th>Encryption Block ID</th>
<th>Representative Interval</th>
</tr>
</thead>
<tbody>
<tr>
<td>U84573</td>
<td>[0.16, 0.2], 0.55, 0.596</td>
<td>1</td>
<td>[0.16, 0.2]</td>
</tr>
<tr>
<td>patient</td>
<td>[0.14, 0.46], 0.54, 0.86</td>
<td>2</td>
<td>[0.393, 0.439]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>[0.55, 0.596]</td>
</tr>
</tbody>
</table>

(a) DSI Index Table

(b) Encryption Block Table

Figure 4.6: Structural Index

In summary, the structural index at the server consists of the DSI index table and the encryption block table. We propose the following theorem:

**Theorem 4.4.1 (Security of Structural Index):** Let $DS(D, C, SI)$ be a hosted XML database system, where $D$ is an XML database, $C$ is the encrypted version of $D$, and $SI$ is the structural index of $D$. Then $DS(D, C, SI)$ is secure (Definition 4.2.4).

Then we prove the correctness of Theorem 4.4.1. We prove this by induction. We first prove the case when there is only one node-type constraint. Then we prove the case when there are multiple node-type constraints.

- **Base case:** only one node-type constraint

We assume by enforcement of a node-type constraint, $\sigma$, that there are $m$ encryption blocks in $C$, with each encryption block containing $n_i$ leaf nodes that are represented by $k_i$ intervals ($n_i \geq k_i$, $1 \leq i \leq m$). Then we prove that there are $\prod_{i=1}^{m} (n_i-1)$ - 1 database systems $DS'(D', C', SI')$ such that $DS'$ satisfies conditions (1) - (4) in Definition 4.2.4. Let $B_i$ be the $i$th encryption block that is obtained by encrypting the subtree rooted at node $x$. Assume that there are $n_i$ leaf nodes in $B_i$ represented by $k_i$ intervals in the DSI index table, with each interval representing (a group of) $n_i$ leaf nodes. Then there are $(n_i-1)$ assignments of values of $m_i$ ($1 \leq i \leq k_i$) such that $\sum_{i=1}^{k_i} m_i = n_i$. Each assignment represents a possible structure of the subtree. For each possible subtree, we assign the same values to those $n_i$ leaf nodes at the same position, i.e., the value $v$ at the $i$th leaf node in the true database is assigned to the $i$th leaf node in the candidate database. For example, for $n_i = 7$ and $k_i = 3$, we can have...
7 = 1 + 1 + 5, = 1 + 2 + 4, = 2 + 3 + 2, \ldots, - 15 possible assignments to assign 7 leaf nodes to 3 intervals (Figure 4.7 shows three such possibilities), with each representing a possible subtree rooted at node A. Since for every encryption block, we have \( \binom{n_i - 1}{k_i} \) possible candidates, and there are \( m \) such encryption blocks, the total number of candidate databases is \( \Pi_{i=1}^{m} \binom{n_i - 1}{k_i - 1} \).

We prove that those candidate databases satisfy conditions (1) - (4) in Definition 4.2.4.

Condition (1) of Definition 4.2.4: As we have shown, the DSI intervals of these candidate databases are identical. Therefore \( SI = SI' \).

Condition (2) of Definition 4.2.4: Since the encryptions of those possible subtrees \( D' \) have the same size and the same occurrence frequency, they are indistinguishable from the true hosted database \( D \) by either the size-based attack or the frequency-based attack. Thus \( D \sim D' \).

Condition (3) of Definition 4.2.4: Out of these candidates, only one has the same structure as the true hosted database \( D \). Thus for every candidate \( D' \), \( D' \neq D \), \( D' \neq A \), where \( A \) is a node-type constraint.

Condition (4) of Definition 4.2.4: For every query \( A \) captured by any security constraint in \( \Sigma \) and its translated query \( A_s \), \( A_s \) will be answered with the structural index indicating the satisfying encryption blocks. Since these candidate databases have the same structural index (as proven in Condition (1) above), \( A_s(S(D), \mu(D, S(D))) = A_s(S(D'), \mu(D', S(D'))).\)

Following the four conditions, for \( \Pi_{i=1}^{m} \binom{n_i - 1}{k_i - 1} - 1 \) fake candidate databases (\( n_i \) is the number of leaf nodes that are represented by \( k_i \) intervals), although \( n_i \) is decided by the input hosted database, since we control \( k_i \) so that \( \Pi_{i=1}^{m} \binom{n_i - 1}{k_i - 1} > K + 1 \), \( \Pi_{i=1}^{m} \binom{n_i - 1}{k_i - 1} - 1 \) must be larger than \( K \), i.e., the database system is secure with the presence of the structural index when there is only one node-type constraint. Note that although the presence of structural index makes the system fail to meet the perfect security requirement, the attacker still cannot tell any fake candidate database from the true one.

- **Induction:** more than one node-type constraint

We assume for \( n \) node-type constraints \( \Sigma = \{\sigma_1, \ldots, \sigma_n\} \) there are \( N \) databases that satisfy Definition 4.2.4. Then we prove that for \( n + 1 \) node-type constraints \( \Sigma' = \Sigma \cup \{\sigma_{n+1}\} \), the hosted database system \( DS(D, C', SI') \) still satisfies Definition 4.2.4, where \( C' \) is the encrypted version of \( D \) with regard to \( \Sigma' \), and \( SI' \) is the DSI index that is constructed with regard to \( \Sigma' \), such that they satisfy the conditions in Definition 4.2.4. We assume by enforcement of \( \sigma_{n+1} \) that the subtrees rooted at node \( x \) are encrypted. If those subtrees have been encrypted by enforcement of \( \Sigma \), i.e., \( \sigma_{n+1} \) has been subsumed by one of the associations in \( \{\sigma_1, \ldots, \sigma_n\} \), the \( N \) candidate databases must be exactly the same as that of \( \Sigma' \). Otherwise, we assume there are \( m \) new encryption blocks in \( C' \) that correspond to the subtrees rooted at node \( x \), with each encryption block containing \( n_i \).
leaf nodes that are represented by $k_i$ intervals $(n_i \geq k_i, 1 \leq i \leq m)$. By using the same reasoning in the base case, we can construct $\prod_{i=1}^{m} \binom{n_i - 1}{k_i - 1}$ candidates $P$ for the subtree rooted at node $x$. Then we construct the candidate databases with regard to $\Sigma'$ in the following manner. For every candidate database out of the $N$ databases, we construct $\prod_{i=1}^{m} \binom{n_i - 1}{k_i - 1}$ candidate databases by replacing the subtree rooted at node $x$ with one possible candidate out of $P$. Therefore, we have $N \times \prod_{i=1}^{m} \binom{n_i - 1}{k_i - 1}$ candidate databases in total with regard to $\Sigma'$. Since $N$ satisfies that $N > K$, and $\prod_{i=1}^{m} \binom{n_i - 1}{k_i - 1}$ is a positive integer that is greater than 1, $N \times \prod_{i=1}^{m} \binom{n_i - 1}{k_i - 1}$ must be greater than $K$. Furthermore, the DSI intervals of the possible candidate databases are identical (Condition (1) of Definition 4.2.4). The encryption of those possible subtrees have the same size and the same occurrence frequency (Condition (2) of Definition 4.2.4). Out of those candidates, there is only one true database that contains the sensitive nodes (Condition (3) of Definition 4.2.4). For any query that is captured by any node-type constraint, these candidates return the same set of encryption blocks as the true database (Condition (4) of Definition 4.2.4). Therefore, the database system is secure with the presence of the structural index with more than one node-type constraint.

In summary, the database with a structural index present satisfies the security definition.

### 4.4.3 Value Index

For value indexing, we choose B-trees so as to support range queries. Each data entry of the B-tree will be of the form $<\text{evalue}, \text{Bid}>$, where evalue is the encrypted version of the value from the database, and Bid is the ID of the encrypted block containing an occurrence of evalue.

Simply encrypting the data values that we want to index on and storing the ciphertext in the B-tree will result in encrypted values following the same distribution as the original plaintext values, opening the door to frequency-based attacks. To mitigate this problem, we advocate “splitting” each plaintext value into one or more ciphertext values in such a way that regardless of the original distribution, the target distribution remains flat, i.e., uniform. Splitting is not enough to guard against frequency-based attack (see Example 4.4.3). Thus, we further use “scaling” on the split data so that the attacker cannot uniquely crack the identity of ciphertext values with the aid of the data frequency knowledge. Unfortunately, “splitting and scaling” is still not secure enough (see Example 4.5.1). Thus, we add noise after splitting and scaling. The details of splitting, scaling and adding noise will be discussed in the next section. Note that although splitting and scaling with noise and block encryption in CBC mode (Section 4.3.1) have the same effect in that they change the distribution of encrypted values, they are different techniques: the splitting&scaling with noise technique is applied on the metadata, while the block encryption in CBC mode is applied on the database.
Agrawal et al. [8] proposed a method of designing order-preserving encryption functions such that the distribution of the encrypted values follows an arbitrarily chosen target distribution, regardless of the input distribution. However, the input distribution considered in their work is a combination of histogram-based and parametric techniques, while we consider a more conservative model where the attacker knows the exact occurrence frequencies of domain values. To protect the data against frequency-based attack, we map the same plaintext values to different ciphertext values, with the target domain size becoming larger than the input domain size. Our goal is that regardless of the distribution of the plaintext values, the distribution of the ciphertext values is always uniform. The naive method is to encrypt the data values in the way that every ciphertext value is a unique number of occurrence. However, since such method requires that every data value is encrypted as a distinct value by using a unique key, it will result in expensive cost of key management and huge size of the value index. Thus we study the problem of how to encrypt the data values in the value index so that minimal number of keys are required for encryption, while the distribution of the ciphertext values is uniform. We describe our approach next.

### 4.4.3.1 Order-preserving Encryption with Splitting, Scaling and Noise

![Frequency of Unencrypted Data Value](image1)  
![Frequency of Encrypted Data Value](image2)

**Figure 4.8:** Data Distribution before Encryption & after Encryption. X axis: Distinct (plaintext/ciphertext) Values. Y axis: Number of Occurrence

**Splitting**

Let \( A \) be an attribute or atomic element being encrypted. Suppose there are \( k \) distinct values \( \{v_1, ..., v_k\} \) in the (active) domain whose numbers of occurrences are \( \{n_1, ..., n_k\} \). Without loss of generality, suppose \( v_1 < v_2 < ... < v_k \) (alphabetical ordering is used for categorical domain). To map each \( v_i \) to multiple ciphertext values so that they have a similar number of occurrences, we look for 3 consecutive positive integers, \( m-1, m, \) and \( m+1 \), such that each \( n_i \) can be expressed in the following form: \( n_i = k_1^i \times (m-1) + k_2^i \times m + k_3^i \times (m+1) \), where \( k_1^i, k_2^i, \) and \( k_3^i \) are non-negative integers. The intuition is to map a data value...
with \( n_i \) occurrences to \( k_1^i + k_2^i + k_3^i \) distinct encrypted values (by using \( k_1^i + k_2^i + k_3^i \) encryption keys), each with \( m - 1, m \) or \( m + 1 \) occurrences. Thus in the distribution of the encrypted values, the frequency of each value will be nearly uniform for the server/attacker, which will defend against the frequency-based attack. For example, Figure 4.8(a) shows an input where values follow a somewhat skewed distribution. This distribution is transformed into a nearly flat distribution (Figure 4.8(b)) where every frequency is 6, 7, or 8. In this example, the input value 90 with number of occurrences 34 \( (34 = 1 \times 6 + 4 \times 7 + 0 \times 8) \), is encrypted into \( 1 + 4 + 0 = 5 \) different encrypted values in Figure 4.8(b) by using 5 distinct keys, with each distinct encrypted value having 6 or 7 occurrences.

Notice that the existence of the triple \((m - 1, m, m + 1)\) is always guaranteed, we ignore the case where some values have just 1 occurrence, as this case is dealt with below. Then \((2, 3, 4)\) is a triple using which any positive integer number \( n_i \) \((n_i > 1)\) can be expressed as \( n_i = k_1^i \times 2 + k_2^i \times 3 + k_3^i \times 4 \). We choose the maximum value of \( m \) for which the triple \((m - 1, m, m + 1)\) works, so intuitively the number of keys \((i.e., \ k_1 + k_2 + k_3)\) needed is reduced. Once the triple \((m - 1, m, m + 1)\) is chosen, for \( n \) distinct plaintext values, \( K = \max_{1 \leq i \leq n} \{k_1^i + k_2^i + k_3^i\} \) is the maximum number of keys needed for encryption. For those values \( v_i \) with only 1 occurrence, if the chosen chunk sizes are \((m - 1, m, m + 1)\), we split \( v_i \) into \( m \) values. In this case, the number of keys required would be \( m \), which in general may be larger than \( K \), the maximum number of keys needed as above.

For efficient query processing purpose, a key requirement of this encryption is that the ciphertext values corresponding to different plaintext values should never straddle each other. More precisely, we require \((*)\): for any two values \( v_i < v_j \), and for any ciphertext values \( c_i^m, c_j^m \) associated with \( v_i, v_j \), respectively, it is necessary that \( c_i^m < c_j^m \). We describe a method for achieving such an encryption. Let \( \text{enc} \) denote any order-preserving encryption function, such as was proposed by [8]. Choose \( k \) distinct random numbers \( w_1, \ldots, w_k \in (0, 1/(k + 1)) \). Without loss of generality, let \( w_1 < w_2 < \cdots < w_k \). Let there be \( n \) distinct plaintext values ordered as \( v_1, v_2, \ldots, v_n \). Let \( \delta = \max_{1 \leq i \leq n} (v_{i+1} - v_i) \). Let \( n_i \) be the number of occurrences of value \( v_i \) in the original database. Recall that out of the \( n_i \) occurrences, chunks of size \( m - 1, m, \) or \( m + 1 \) need to be mapped to a given encrypted value. Then we transform the first chunk of occurrences of \( v_i \) to \( \text{enc}(v_i + w_1 \delta) \). More generally, we map the \( n \)-th chunk of occurrences to \( \text{enc}(v_i + (\sum_{1 \leq j \leq n} w_j) \delta) \). In other words, the \( n \)-th chunk is displaced from \( v_i \) by a fraction of the gap \( \delta \) given by the sum \( w_1 + \cdots + w_n \). The result is encrypted using an order-preserving encryption function. It is straightforward to show that the resulting encryption satisfies the required condition \((*)\) above. Splitting of a ciphertext value \( c \) into multiple values can be easily implemented by adding randomly chosen low-order bits to \( c \). Example 4.4.2 illustrates the splitting approach.

**Example 4.4.2 [Splitting]**
Suppose two successive values are 23 and 32. We have determined \( k = 4 \). The chunk sizes are 5, 6, 7. The gap is \( \delta = 32 - 23 = 9 \). Suppose that the chosen random numbers are \( w_1 = 0.05, w_2 = 0.1, w_3 = 0.05, w_4 = 0.02 \). Then 23 would
be displaced to each of the values $-23 + 0.05 \times 9 = 23.45$, $23.45 + 0.1 \times 9 = 24.35$, $24.35 + 0.05 \times 9 = 24.80$, $24.80 + 0.02 \times 9 = 24.98$. The displaced values would then be encrypted using some order-preserving encryption function. Assume the number of occurrence of 23 is 25. Since $25 = (1 \times 5) + (1 \times 6) + (2 \times 7)$, the split value 23.45 is of number of occurrence 5, 24.35 is of number of occurrence 6, and both 24.80 and 24.98 are of number of occurrences 7. The value 32 would be handled in a similar fashion.

If the domain is not real or rational, then we map it to a domain that is. The client keeps the mapping between categorical values and natural numbers.

### Scaling

A key property of the splitting technique described so far is that the total number of occurrences of any value is unchanged by the transformation. That is, $\sum_{1 \leq j \leq k} n_j = \sum_{1 \leq j \leq k} f_j$, where $n_j$ ($f_j$) are the occurrence frequencies of plaintext (ciphertext) values, and $k$ ($\ell$) is the number of distinct plaintext (ciphertext) values. An attacker with an exact knowledge of occurrence frequencies of input values can group adjacent ciphertext values together until they match a particular occurrence frequency of an input value. Example 4.4.3 shows such an attack.

#### Example 4.4.3 [Example of Insecurity of Splitting Alone]

Assume the attacker knows the plaintext value 23 is of number of occurrence 25. Then based on the splitting result of Example 4.4.2, the attacker can group the first four encrypted values (23.45, 24.35, 24.80, and 24.98) together and map them to value 23, since their numbers of occurrences sum up to 25.

We use scaling to defend against such attacks. Let $s_i$ be a scale factor. We typically want to use a small real number in the range $[1, 10]$, since the index size is affected by the scale factor. When we split a value $v_i$ into ciphertext values $c_1^i, \ldots, c_n^i$, we replicate each index entry corresponding to $c_j^i, s_i$ times, so that the number of occurrences of each ciphertext value is the same. Note that all $s_i$ replicas of the index entry for $c_j^i$ will point to different encryption blocks. Such pointing is fake. The scale factor $s_i$ is chosen independently for each value $v_i$. Therefore, without knowing the scaling factor used, it is not possible for the attacker to crack the identity of ciphertext values.

### Noise + Splitting & Scaling

By the order-preserving requirement, even after splitting and scaling, the mapping between the smallest plaintext and ciphertext values, and that between the largest plaintext and ciphertext values, are still preserved. More seriously, simply applying splitting and scaling may result in insecure query answering (see Example 4.5.1 in Section 4.5.3). To address these two security issues, we add noise to the metadata after splitting and scaling. Note that after splitting, all plaintext values and their corresponding ciphertext values are ordered. Then for any two adjacent values $v_i$ and $v_j$, which are split into $\{c_1^i, c_2^i, \ldots, c_n^i\}$ and $\{c_1^j, c_2^j, \ldots, c_n^j\}$, where $c_1^m < c_j^m$, we add values $\{f_1, f_2, \ldots, f_k\}$ between $c_1^m$ and $c_j^m$ such that $c_1^m < f_1 < f_2 < \ldots < f_k < c_j^m$. We call those fake values added
between \( c_i^m \) and \( c_i^1 \) the noise of value \( v_i \). Furthermore, for the smallest plaintext value \( v_1 \) and the largest plaintext value \( v_n \), we add noise before the first (i.e., the minimum) split value of \( v_1 \) and after the last (i.e., the maximum) split value of \( v_n \). Thus, by adding noise at the head and the tail, the mapping between the smallest plaintext and ciphertext values, and between the largest plaintext and ciphertext values, will no longer be revealed. Example 4.4.4 illustrates the idea of noise. Note that for any value \( v_i \), its noise will point to the same encryption block IDs as its own in the metadata.

**Example 4.4.4 [Noise]**

Assume there are three distinct plaintext values 1, 2 and 3. Value 1 is split into 10, 11, value 2 is split into 200, 201, 202, and value 3 is split into 3000 and 3001. We add the following noise:

1. We add values 6, 7 before 10.
2. We add values 70, 80 and 90 between 11 and 200 as noise of value 1.
3. We add values 500, 1000 and 2000 between 202 and 3000 as noise of value 2.
4. We add values 5000 and 5100 after 3001 as noise of value 3.

The noise added between different pairs of ciphertext values may be of different sizes. However, we require that: (1) after splitting and noise being added, the number of distinct ciphertext values \( n \) and the number of distinct plaintext values \( k \) satisfies that \((\binom{k-1}{n-1}) > K + 1\), where \( K \) is a given security threshold in Definition 4.2.4, and (2) for any distinct plaintext value \( v_i \) that is split into \( k_i \) distinct values and with \( f_i \) distinct fake noise values added, the size of noise \( f_i \) satisfies that \((2^{k_i + f_i} - 1) > 1/\gamma\), where \( \gamma \) is a given security threshold in Definition 4.2.5. The reason for this constraint will be discussed in Section 4.5.3.

After we add noise values to the metadata, we scale their number of occurrences to be equal to that of the split values. Therefore by observing the distribution of the data values in metadata, the attacker cannot tell whether it is noise or real split values. We propose the following theorem:

**Theorem 4.4.2 (Security of Value Index):** Let \( DS(D, C, VI) \) be a database system, where \( D \) is an XML database. Let \( C \) be the encrypted version and \( VI \) be the value index of \( D \). Then \( DS(D, C, VI) \) is secure (Definition 4.2.4).

**Proof:** Let \( \Sigma \) be a given set of security constraints. We first prove that for one specific association constraint, the theorem is correct. Then we induce on the number of association constraints and prove the correctness of the theorem for a set of association constraints.

- **Base case:** only one association-type constraint:

Assume there is only one association-type security constraint //\( p : (q_1, q_2) \in \Sigma \); without loss of generality, we assume that every value of \( p/q_1 \) is encrypted. Suppose there are \( k \) distinct plaintext values and \( n \) distinct
ciphertext values of attribute/element \( p/q_1 \) in \( D \). Note that \( n > k \). We prove that there are at least \( K \) (defined in Definition 4.2.4) databases where \( v_i \in \{v_1, ..., v_k\} \) and \( y \in \text{dom}(p/q_2) \), such that \( \langle /p : [q_1 = v_i][q_2 = y] \rangle \) does not hold for any of them.

For \( k \) distinct plaintext values \( \{v_1, ..., v_k\} \) and \( n \) distinct ciphertext values \( \{c_1, ..., c_n\} \), the attacker tries to map \( c_i \) with \( v_j \). The number of mappings together with appropriate scalings is determined by the number of ways of partitioning the set \( \{c_1, ..., c_n\} \) into \( k \) non-empty subsets \( (S_1, ..., S_k) \), in an order-preserving way, i.e., for every \( v_i \) in \( S_i \) and for every \( v_j \) in \( S_j \) (\( i \neq j \)), we have \( v_i < v_j \). There are \( \binom{n-1}{k-1} \) possible partitions.

For those \( \binom{n-1}{k-1} \) candidate databases \( D' \), we show that they satisfy the four conditions of Definition 4.2.4:

Condition (1) of Definition 4.2.4: As we have shown, these candidate databases have the same value index. Therefore \( VI = VI' \).

Condition (2) of Definition 4.2.4: Since the encryptions of those possible subtrees \( D' \) have the same size and the same occurrence frequency, they are indistinguishable from the true hosted database \( D \). Thus \( D \sim D' \).

Condition (3) of Definition 4.2.4: Out of those candidates, there is only one database \( D \) that has the same value assignments as the true hosted database. Thus \( D \models A \), where \( A \) is an association-type constraint. However, for the other candidate databases \( D' \neq D \), \( D' \not\models A \).

Condition (4) of Definition 4.2.4: For every query \( A \) captured by any security constraint in \( \Sigma \) and its translated query \( A_s \), \( A_s \) will be answered with the value index indicating the satisfying encryption blocks. Since these candidate databases have the same value index (as proven in Condition (1) above), \( A_s(S(D), \mu(D, S(D))) = A_s(S(D'), \mu(D', S(D'))) \).

Following the above proof, out of \( \binom{n-1}{k-1} \) candidate databases, there is only one that is the true hosted database. Note that here \( n \) is the number of distinct ciphertext values, and \( k \) is the number of distinct plaintext values. Although \( k \) is decided by the input hosted database, since we always control \( n \) by splitting and adding noise, so that \( \binom{n-1}{k-1} > K + 1 \), where \( K \) is a given security threshold in Definition 4.2.4, the number of fake candidate databases \( \binom{n-1}{k-1} - 1 \) is always greater than \( K \). Thus the database system is secure with the presence of the value index when there is only one association-type constraint available. Note that although the presence of value index makes the system fail to meet the perfect security requirement, the attacker still cannot tell any fake candidate database from the true one.

**Induction:** more than one association-type constraint:

Assume for \( n \) association constraints \( \Sigma = \{\sigma_1, \sigma_2, ..., \sigma_n\} \), \( DS(D', C, VI) \) is secure with regard to \( \Sigma \). Then for \( \Sigma' = \Sigma \cup \{\sigma_{n+1}\} \), and \( DS(D, C, VI) \),
Chapter 4. Query Evaluation over Encryption XML Databases

VI'), where \( C' \) is the encrypted version of \( D \) by considering \( \Sigma' \), and \( VI' \) is constructed with regard to \( \Sigma' \), we prove that DS\((D, C', VI')\) is secure.

We assume that for DS\((D, C, VI)\), there are \( N \) candidate databases that satisfy Definition 4.2.4. We also assume \( \sigma_{n+1} \) is of the form \( p : (q_1, q_2) \), which is an association-type constraint. Without losing generality, we assume that to enforce the constraint \( \sigma_{n+1} \), every value of \( p/q_1 \) is encrypted. We consider two cases:

- Case 1: neither \( p/q_1 \) nor \( p/q_2 \) of \( \sigma_{n+1} \) is involved in any association constraint in \( \sigma_1, \ldots, \sigma_n \). By using the same reasoning for candidate databases as in the base case, we can conclude that for the query \( \langle /p : [q_1 = v_i][q_2 = y_j] \rangle \), the attacker has \( T = \binom{n-1}{k-1} \) choices for the cipher values of \( p/q_1 = v_i \) to be mapped to, where \( n \) and \( k \) are the number of distinct plaintext and ciphertext values of \( q_1 \) that satisfy the \( p/q_1 \) expression. Then we can construct the candidate databases for \( \Sigma' = \Sigma \cup \{\sigma_{n+1}\} \) as follows: by using the \( N \) candidate databases as the basis, for each candidate database in this set, we construct \( T \) new databases in which the cipher values of \( p/q_1 = v_i \) are assigned with one possible mapped plaintext value, while the other data values that are of different elements/attributes are kept unchanged. Thus we have \( N \times T \) candidate databases for \( \Sigma' = \Sigma \cup \{\sigma_{n+1}\} \). It is straightforward that the value index of the possible candidate databases are identical (Condition (1) of Definition 4.2.4). The encryption of those possible subtrees have the same size and the same occurrence frequency (Condition (2) of Definition 4.2.4). Out of those candidates, there is only one true database that contains the sensitive associations (Condition (3) of Definition 4.2.4). For any query that is captured by any node-type constraint, these candidates return the same set of encryption blocks as the true database (Condition (4) of Definition 4.2.4). Since \( N \) satisfies that \( N > K \), and \( T \) is a positive integer that is greater than 1, \( N \times T \) must be greater than \( K \). Therefore, the database system is secure with the presence of the value index when there are more than one association-type constraint.

- Case 2: at least one of the elements \( p/q_1 \) and \( p/q_2 \) of \( \sigma_{n+1} \) are involved in some association constraint in \( \sigma_1, \ldots, \sigma_n \). Without losing generality, we assume that \( p/q_1 \) is involved in some association constraint and that every value of \( p/q_1 \) is encrypted correspondly. Then in those \( N \) candidate databases for \( n \) associations, the plaintext value of every ciphertext value of \( p/q_1 \) that is mapped to is fixed. Therefore, if every value of \( p/q_2 \) is not encrypted or \( p/q_2 \) is also involved (i.e., the mapping of plaintext and ciphertext values of \( p/q_2 \) is fixed), there are at least \( N \) candidate databases for \( \Sigma' = \Sigma \cup \{\sigma_{n+1}\} \). Otherwise, there are \( N \times T \) candidate databases for \( \Sigma' \), where \( T \) is the number of possible mappings of plaintext and ciphertext values of \( p/q_2 \). The proof of four conditions are similar to Case 1.
Since Definition 4.2.4 is satisfied, DS(D, C, VI) is secure.

Note that by using different encryption techniques for the database and for the value index, we can ensure their encrypted value sets are disjoint, so that by accessing all of them, the attacker cannot infer any useful information he or she otherwise could not.

This theorem shows that by collecting the distribution of the encrypted data from the B-tree index, the server cannot infer any association that would violate any security constraint \( p : (q_1, q_2) \in \Sigma \). The price of this protection is that the size of the B-tree index is more than it would be for an unencrypted database. This increase in size is proportional to the scaling used and the noise added. Second, because of splitting, aggregate queries involving COUNT cannot be evaluated without decryption, although queries involving MAX/MIN can still be evaluated correctly without decryption. The details of query processing are discussed in Section 4.5.

We refer to the particular encryption technique described here as order-preserving encryption with splitting, scaling and noise (OPESS).

Combining both the structural index and value index, we propose the following theorem:

**Theorem 4.4.3 (Security of DSI and Value Index)**: Let \( D \) be an XML database, \( C \) be the encrypted version, \( SI \) be the DSI index of \( D \), and \( VI \) be the value index of \( D \). Then DS(D, C, SI \cup VI) is secure.

**Proof**: Let \( \Sigma_n \) be the set of node-type constraints and \( \Sigma_a \) be the set of association constraints. We assume there are \( M \) candidate databases for \( \Sigma_n \) and \( N \) candidate databases for \( \Sigma_a \). We say that the node-type constraint \( \sigma_1 \) subsumes the association constraint \( \sigma_2 \) if the attributes/elements specified by \( \sigma_2 \) are in the subtree of nodes specified by \( \sigma_1 \). Then if for every \( \sigma \) in \( \Sigma_a \), there exists \( \sigma' \) in \( \Sigma_n \) such that \( \sigma' \) subsumes \( \sigma \), the number of candidate databases for \( SI \cup VI \) is \( M \), which is the number of candidate databases for \( \Sigma_n \). Otherwise, the number of candidate databases for \( \Sigma_n \cup \Sigma_a \) is at least \( M \), since for every database out of \( M \) candidate databases for \( \Sigma_n \), we can replace the values in the database that correspond to some un-subsumed association constraints with any possible mapping that is represented by one candidate out of \( N \) candidate databases. Since \( M \) satisfies that \( M > K \), the number of candidate databases for \( SI \cup VI \) must be greater than \( K \). The proof that these candidate databases that satisfy the four conditions in Definition 4.2.4 is similar to the proofs of Theorem 4.4.1 and 4.4.2.

**Updates**

When there are insertions/deletions on the database, the value index has to be updated accordingly. There are two issues for updates on value index: efficiency and security. To ensure security, the frequency occurrence of every data value must be the same. To achieve this goal, the first approach is to apply scaling so that by replication data values are of the same number of occurrence. Although this approach supports efficient incremental maintenance on value
index, it may incur large space due to replicated values. Another approach is to apply splitting, scaling, and adding noise on the updated data, which may be of expensive computation cost but cheaper space cost compared with the first approach. Developing a secure metadata for efficiently supporting updates (for both computation and space concerns) is a open problem.

### 4.5 Query Processing

The major steps in query answering are as follows: (1) The client translates the query $Q$ into $Q_s$ by replacing the tags and data values individually with their encrypted format, whenever the tag/value was encrypted in $\text{enc}(D)$; (2) the server labels each (encrypted) query node with its DSI index entry retrieved from the DSI index table and computes structural joins, which prune index entries at query nodes. After the pruning, the remaining indices represent the nodes that satisfy the structural predicates of the query; (3) for the value-based constraints in the query, the server looks up the B-tree and returns the intervals (index entries) that represent the encryption blocks that meet the value constraints; (4) the server then "joins" the result of Step 2 with the results of Step 3 and obtains a set of unencrypted nodes, or a set of block IDs if the output node is encrypted in some blocks. The corresponding nodes or the encrypted blocks satisfy both structural and value constraints of the query and are returned to the client; (5) the client decrypts the received encrypted blocks and applies the original query $Q$ on the remaining decrypted result. These steps are elaborated upon below. We use the query shown in Figure 4.9 as a running example.

#### 4.5.1 Query Translation at the Client

We encrypt the tags and value constraints individually in the query while still preserving the query structure. 

**Translation of query tags**

Recall that the constraint graph (in Section 4.3.2) records the nodes that are encrypted. For any query node that has the same tag as one of those nodes, its
tag will be encrypted by the Vernam cipher scheme [46] with the same keys used for the construction of the DSI index table. The structural axis in the query, such as child (/), descendant (//), will not be modified.

**Translation of Value-based constraints**

If the query node has a value-based constraint, the value will be encrypted by the same encryption function and the same keys as used for the construction of the B-tree index. We make use of the order-preserving property of OPESS functions in translating value constraints in the query. The details of the transformation are shown in Figure 4.10. Basically, $E(v_1 + w_1 \cdot \delta)$ returns the smallest value of the split value of $v_1$, and $E(f)$ returns the largest value (including noise) of the split value of $v_1$. Since for any value $v_i$, its noise points to the same encryption block IDs as its own in the metadata, such translation will not result in irrelevant encryption blocks that will be returned by the server. Figure 4.9 shows an example of query translation on the client based on the instance shown in Figure 4.2. In this example the patient node is kept unencrypted, while the Insurance and coverage nodes are replaced with their encrypted formats.

### 4.5.2 Query Translation at the Server

After receiving the query $Q_s$ from the client, the server answers the query by the following steps:

**Step 1: Translation of Query Structure:** The server first obtains index entries associated with each query node type by consulting the DSI index table. Note that some nodes may have a tag that is encrypted. For example, the intervals [0.16, 0.2] & [0.55, 0.596] will be returned by retrieving "U84573" (the encrypted value of query node “SSN”) from the DSI index table (Figure 4.6(a)). We call the interval(s) of the distinguished (i.e., query answer) node distinguished interval(s).

Once query nodes have lists of intervals associated with them, the server computes any of the standard structural join algorithms (see Section 2.1.3) to prune away intervals that do not match structural constraints of the query. The remaining intervals represent tuples of nodes that satisfy the structural constraints of the query. For our running example, [0.16, 0.2] & [0.55, 0.596]
will be returned.

Step 2: Translation of Value-based Constraints: Each value-based constraint can be treated as a triple \(<\text{tag}, \text{op}, \text{value}>\), where \(\text{op} \in \{<, >, \leq, \geq, =\}\). Here \(\text{tag}\) and \(\text{value}\) are both in encrypted format. Each value-based constraint is translated as a range query, which is necessary because of the splitting that was applied before encryption. Figure 4.9 shows how the value-based constraint is translated into a corresponding range query on the B-tree index. The server retrieves the index entries from a B-tree index, corresponding to entries that match the triple. Recall that these index entries correspond to encrypted blocks. Therefore, for the value-based constraint \(<\text{coverage}>100000\), the server retrieves the entry pointing to Block 2.

Step 3: Obtaining Final Results: Step 1 yields index entries (intervals) corresponding to the distinguished node of the query, while Step 2 yields index entries associated with encrypted blocks. In this final step, the server computes a final set of structural joins to determine the subset of encryption block id’s that contain the distinguished node intervals. The resulting set of block ID’s are used to fetch the blocks from the encrypted database \(\text{enc}(D)\), which are then shipped to the client. In our example, block 1 will be returned to the client.

4.5.3 Security of Query Answering

We have shown that our encryption scheme is secure (Theorem 4.3.1). We also have shown that by adding metadata on the server side, the metadata is secure (Theorems 4.4.1 and 4.4.2). The next scenario to consider is an attacker being able to observe client queries and server responses. Could the attacker eventually learn enough about the information to believe it is sensitive? Before we show the security result, we discuss our observation. We observe that the scheme of splitting & scaling without noise will result in insecure query answering on the server. We use Example 4.5.1 to illustrate this observation.

Example 4.5.1 [Insecurity of Query Answering against Splitting & Scaling without Noise]

Assume that there are two attributes \(x\) and \(y\), in the database. To protect the association between them, all values of attribute \(x\) are encrypted. There are three distinct plaintext values for attribute \(x\): 1, 2 and 3. In the metadata, value 1 is split into 10 and 11, value 2 is split into 200, 201, 202, and value 3 is split into 3000 and 3001. There is no noise added.

Assume that there are three queries in a sequence, all of which are specified on the associations between the values of attribute \(x\) and attribute \(y\). The queries are \(Q_1: p[\@x=1][\@y=1]\), \(Q_2: p[\@x=2][\@y=1]\), and \(Q_3: p[\@x=3][\@y=1]\). Note that value 1 of attribute \(y\) is not encrypted. The client translates the queries as follows and sends them to the server:

1. \(Q'_1: p[10 \leq @x \leq 11][@y=1]\) (corresponding to \(@x=1\) originally)
2. \(Q'_2: p[200 \leq @x \leq 202][@y=1]\) (corresponding to \(@x=2\) originally)
3. \(Q'_3: p[3000 \leq @x \leq 3001][@y=1]\) (corresponding to \(@x=3\) originally)


We assume that the attacker knows it is an order-preserving encryption scheme. Therefore, he or she knows that the lower bound and upper bound of each range in the query must correspond to the lower bound and upper bound of a split value. Those upper and lower bounds must follow exactly the same order as their plaintext values. From the three received queries $Q'_1$, $Q'_2$, and $Q'_3$, the attacker observes that there is no overlap between the three ranges in these queries; thus, these three ranges must map to three distinct values. By prior knowledge, the attacker knows there are three distinct plaintext values 1, 2 and 3 of attribute $x$. Therefore, he/she can easily derive the following mapping:

1→10, 11, 2→200, 201, 202 and 3→3000, 3001 and successfully cracks the identification of the split ciphertext values of attribute $x$. Consequently, by observing the answers of these three queries, the attacker’s probability of whether the sensitive association between values of attribute $x$ and $y$ exists becomes 1.

In other words, the query answering procedure is not secure!

We now prove that our query-answering procedure is secure after noise is added to the value index.

**Theorem 4.5.1 (Security of Query Answering)**: Query answering on the server is secure (Definition 4.2.5).

**Proof of Theorem 4.5.1**

Let $\Sigma$ be a set of security constraints. Let $A$ be any query captured by a security constraint. Given a set of encryption blocks $\{B_1, \ldots, B_n\}$ in the encrypted database, as defined in Definition 4.2.5, we use $B_i(A)$ to denote that the block $B_i$ satisfies the query $A$. Then we prove the correctness of Theorem 4.5.1 for two cases separateley: $A$ is a structural-type constraint and $A$ is an association-type constraint. Specifically, we prove that by observing $q$ queries, the attacker's belief probability $Bel_\gamma(B_i(A))$ is always less than the given security threshold $\gamma$ (defined in Definition 4.2.5).

**Query answering regarding node-type constraint $A$:**

Let $D$ be the plaintext database and $S(D)$ be its encrypted version. Let $A$ be any query captured by a node-type security constraint. We want to show that by observing queries and responses, the attacker's belief probability $Bel_\gamma(B_i(A))$ is always less than the security threshold $\gamma$. As we have shown in the proof of Theorem 4.4.1, for every encryption block $B_i$ that contains $n_i$ leaf nodes, which are represented as $k_i$ intervals in the structural index, the number of possible structure trees of $B_i$ that the attacker can construct is $\binom{n_i-1}{k_i-1}$, and these $\binom{n_i-1}{k_i-1}$ candidates are equally likely to satisfy $A$ to the attacker. Thus before observing any query, the attacker's belief $Bel_\gamma(B_i(A))$ is $1/\binom{n_i-1}{k_i-1}$. Recall that we chose the Vernam cipher to encrypt queries. Based on the perfect security property of the Vernam cipher (i.e., for any plaintext tag $M$ and the ciphertext tag $C$, $\text{Prob}(M=M) = \text{Prob}(M=M|C=C)$), the (translated) queries and responses will not increase the attacker's belief probability of which structure tree satisfies $A$. Thus by observing $q$ queries and their responses, the attacker's belief probability $Bel_\gamma(B_i(A))$ is still $1/\binom{n_i-1}{k_i-1}$. As we have shown in Section 4.4.2, we control the
number of intervals \( k_i \) in the structural index so that \( \binom{\nu - 1}{k_i - 1} > 1/\gamma \). Thus the attacker’s belief probability \( \text{Bel}_q(B_i(A)) \) is always less than \( \gamma \), where \( \gamma \) is the given security threshold in Definition 4.2.5. Therefore, regarding node-type constraints, query answering is secure.

**Query answering regarding association-type constraint** A

For the reasoning of whether query answering reveals any sensitive information of any association-type constraints, we need to consider the queries that contain value-based constraints, which may help the attacker to crack the identification of ciphertext values, and consequently increase the belief probability \( \text{Bel}_q(B_i(A)) \). The following gives more details of how the attacker increases his/her belief by observing queries. Recall that by our OPESS scheme, every plaintext value \( v \) is mapped to a set of ciphertext values \( \{c_1, \ldots, c_n\} \) (in ascending order). We consider \( \{c_1, \ldots, c_n\} \) as the cover of value \( v \) and we call \( c_1 \) and \( c_n \) the starting point and ending point of the cover. We assume that the attacker knows that by our query answering procedure, every value-based constraint in the query is translated into a range with a lower bound \( a \) and an upper bound \( b \) (e.g., \( x=1 \) is translated to \( 10 < x < 11 \)), where \( a \) corresponds to the starting point of a plaintext value \( v_1 \), and \( b \) corresponds to the ending point of a plaintext value \( v_2 \) (\( v_1 \) and \( v_2 \) may be the same). Then by observing a sequence of ranges in the queries, he/she will try to map the lower bounds and upper bounds of the ranges to the starting and ending point of the covers. Specifically, he/she sorts the lower bounds and upper bounds of the ranges and from the sorting result, he/she determines a set of non-overlapping ranges in ascending way. For example, for two ranges \( 10 < @x < 200 \) and \( 20 < @x < 3000 \), the attacker can have three non-overlapping ranges: \([10, 20), [20, 200), (200, 3000]\). For every non-overlapping range, the attacker believes that the ciphertext values in it map to the same plaintext value. Thus the more queries and responses the attacker observes, the more non-overlapping ranges he/she can have, which may help him/her to crack the identification of ciphertext values and increase \( \text{Bel}_q(B_i(A)) \). In the next, we prove that although \( \text{Bel}_q(B_i(A)) \) is increasing by observing queries and answers, the attacker’s belief is always less than the given security threshold \( \gamma \).

Without losing generality, we assume the sensitive association is on the data values of two tags, \( x \) and \( y \). We also assume the data values of \( x \) are encrypted and the data values of \( y \) are left as plaintext. We assume that there exist \( m \) non-overlapping ranges \( R \) from \( n \) ranges of the queries \( \{Q_1, \ldots, Q_n\} \). We assume there are \( N \) distinct plaintext values of the attribute \( x \) in the database. Then we prove that the attacker’s belief is always less than \( \gamma \) by discussion of the following two cases: (1) \( m = N \), i.e., the number of non-overlapping ranges equals the number of distinct plaintext values in the database, and (2) \( m < N \), i.e., the number of non-overlapping ranges is less than the number of distinct plaintext values in the database. Following our model and query answering procedure, \( m > N \) is impossible.

- **Case 1**: \( m = N \), i.e., the number of non-overlapping ranges equals the number of distinct plaintext values. The attacker can infer that these \( m \)
ranges must map to \( N \) distinct values in an order-preserving way. For example, assume the tag \( x \) has three distinct plaintext values 1, 2, 3 and 4. Based on the OPESS scheme in Example 4.4.4, three queries in Example 4.5.1 will be rewritten as: (1) \( Q_1': p[10 \leq @x \leq 90][@y=1] \) (corresponding to \( @x=1 \) originally), (2) \( Q_2': p[200 \leq @x \leq 2000][@y=1] \) (corresponding to \( @x=2 \) originally), and (3) \( Q_3': p[3000 \leq @x \leq 5100][@y=1] \) (corresponding to \( @x=3 \) originally). The attacker observes that there is no overlap between the three ranges in the queries. By prior knowledge, the attacker knows it is an order-preserving scheme and that there are three distinct plaintext values 1, 2 and 3 for attribute \( x \). Therefore, these three ranges must map to three distinct values. Consequently, he/she can derive the mapping: 1 \( \rightarrow \{10, 11, 70, 80, 90\} \), 2 \( \rightarrow \{200, 201, 202, 500, 1000, 2000\} \), and 3 \( \rightarrow \{3000, 3001, 5000, 5100\} \). Assume the attacker knows that noise may exist. After obtaining these three mappings, he/she will guess which values are possibly fake noise and which are true values. The following are some examples of those possible guesses. For 1 \( \rightarrow \{10, 11, 70, 80, 90\} \), the attacker can guess: (1) 1 is split to 10, and values 11, 70, 80, 90 are noise, or (2) 1 is split to 10 and 11, and values 70, 80, 90 are noise, or (3) 1 is split to 10 and 70, and values 11, 80 and 90 are noise, etc.

There are \( 2^5 - 1 \) possible guesses of the mapping schemes for value 1. Similarly, for 2 \( \rightarrow \{200, 201, 202, 500, 1000, 2000\} \), the attacker has \( 2^6 - 1 \) schemes for value 2. For 3 \( \rightarrow \{3000, 3001, 5000, 5100\} \), the attacker has \( 2^4 - 1 \) schemes for value 3. Since there is only one mapping scheme that is correct, the attacker's belief of cracking the identification of value 1 is \( 1/(2^5 - 1) \), similarly \( 1/(2^6 - 1) \) for value 2, and \( 1/(2^4 - 1) \) for value 3.

In general, for the distinct plaintext value \( v_i \), which is split into \( k_i \) distinct values and with \( f_i \) distinct fake noise values added, there are \( (2^{k_i} + f_i) - 1 \) number of candidate mapping schemes. Since when adding noise, we require that \( (2^{k_i} + f_i) - 1 > 1/\gamma \) (in Section 4.4.3), where \( \gamma \) is a given security threshold in Definition 4.2.5, the attacker's belief about cracking the identification of value \( v_i \) is always less than \( \gamma \). Consequently, his/her belief of whether any encryption block contains the sensitive association on value \( v_i \) is always less than a given security threshold \( \gamma \).

- **Case 2**: \( m < N \), i.e., the number of non-overlapping ranges is less than the number of distinct plaintext values. Then the attacker can map each range to either a distinct value or a set of values. For example, based on the OPESS scheme in Example 4.4.4, two queries \( Q_1: p[x \geq 2][y=1] \) and \( Q_2: p[x < 2][y=1] \) will be rewritten as: (1) \( Q_1': p[x \geq 200][y=1] \) (corresponding to \( x \geq 2 \) originally). (2) \( Q_2': p[x < 200][y=1] \) (corresponding to \( x < 2 \) originally). From the two ranges in \( Q_1' \) and \( Q_2' \), there are two non-overlapping ranges: \( x \geq 200 \) and \( x < 200 \). The attacker can map the range \( x \geq 200 \) to 1, 2, or \{1, 2\}. Similarly he/she can map the range \( x < 200 \) to 2, 3, or \{2, 3\}. Then he/she can have the following possible mappings: (1) \{1, 2\} \( \rightarrow \{10, 11\} \), \{3\} \( \rightarrow \{3000, 3001, 5000, 5001\} \), 201-2000 are noise, (2) \{1\} \( \rightarrow \{10, 11\} \), \{2, 3\} \( \rightarrow \{3000, 3001, 5000, 5001\} \), 201
Chapter 4. Query Evaluation over Encryption XML Databases

- 2000 are noise, (3) \{1\} → \{10, 11\}, \{2\} → \{200, 201 - 2000\}, \{3\} → \{3000, 3001, 5000, 5001\}, etc.. Note that the correct one (i.e., the mapping (3)) is included in these schemes. Since mapping (3) alone yields \((2^5 - 1)\) mappings for value 1, \((2^6 - 1)\) for value 2, and \((2^4 - 1)\) for value 3 (the reason is explained in Case 1), the attacker’s belief of the identification of value 1 must be less than \(1/(2^5 - 1)\), similarly less than \(1/(2^6 - 1)\) for value 2, and less than \(1/(2^4 - 1)\) for value 3.

In general, when the number of non-overlapping ranges is less than the number of distinct plaintext values, the attacker can map each range to either a distinct value or a set of values, which will result in a larger space of possible mappings and consequently a smaller probability, compared with the case that mapping each range to a distinct value. Since we already proved in Case 1 that when every range is mapped to a distinct value, the probability is always less than the security threshold \(\gamma\), the probability of Case 2 must also be less than \(\gamma\).

To sum up, regarding both node-type and association-type security constraints, our query answering procedure is secure.

4.5.4 Query Post-Processing

The query post-processing on the client side is done in the following steps:

Step 1: Upon receiving the result blocks \(A\) from the server, the client first decrypts them. For our example, the client decrypts block 1 that is returned by the server. Let \(A'\) be the decryption result of the blocks.

Step 2: If the root of all element trees in \(A'\) is the same as the output node of the original query \(Q\), \(A'\) will be returned as the final answer (our example follows such a case). Otherwise,

Step 3: The client rewrites the original query \(Q\) to be \(Q'\) and applies \(Q'\) to \(A'\). The rewriting is quite simple. Let the root of all element trees in \(A'\) be \(r\). All query nodes that are ancestors of \(r\) in the instance as well as their witness predicates (i.e., \([]\) predicate in query) are removed from \(Q\), while the remaining query nodes and predicates are kept intact. For example, for the query

\[
/\text{Insurance}[/\text{@coverage} > 100000]//\text{policy*},
\]

block 2 will be returned. Since the root of the element in this encryption block is \(\text{Insurance}\), which is ancestor of the query output node \(\text{policy#}\), \(\text{Insurance}\) and its witness predicate \([/\text{@coverage} > 100000]\) will be removed. The rewritten \(Q'\) is

\[
/\text{policy#}.
\]

The client applies the rewritten query \(Q'\) to the decryption result. The answer is the final query of the original query \(Q\). It is straightforward to see that this yields the same answer as \(Q\) applied to the original database \(D\).

A final note is that if the query involves aggregation involving the functions MIN/MAX, we can still evaluate them without decrypting data. However, if it involves other aggregate functions, the values would have to be decrypted by the client and then evaluated. Compared to [8], we are unable to process COUNT queries without decryption. But unlike their scheme, our scheme assumes conservatively that the server/attacker knows the exact occurrence frequencies of
domain values. Unfortunately their approach is not secure regarding our conservative attack model. For join queries, by our approach, the relevant encryption blocks have to be decrypted by the client for post-processing too.

4.6 Activities on Client Side

There are a few activities happened on the client side. First, he/she encrypts the database before he/she sends it to the server. Second, he/she constructs both structural index and value index as the metadata. For these two activities, the client has to store all keys that he/she used. Third, he/she raises and translates the queries with appropriate keys. Fourth, he/she decrypts the returned answers from the server by using appropriate keys, and post-process the queries on the decryption result if it is needed. For these activities, key management is crucial to the client. In this section, we first discuss the key management, specifically the size of keys, on the client side (Section 4.6.1). Second, we discuss the overhead and benefit that the client gets by implementing these activities (Section 4.6.2).

4.6.1 Key Management

There are three kinds of keys that have to be stored on the client side: (1) the keys for encryption of database, (2) the keys for encryption of tags in structural index, and (3) the keys for splitting values in value index. We name these three kinds of keys as key I, key II, and key III.

Size of Key I: Since for encryption on database, we apply block cipher in CBC mode with initial vector IV, the initial vectors that are used must be stored on the client side for decryption. Since different encryption blocks use different initial vectors (and the same key), the size of key I is decided by number of encryption blocks in the database. Formally, assume the initial vector is an $k$-string, $|Key I| = k \times \#$ of encryption blocks.

Size of Key II: We apply one-time-pad encryption to encrypt the tags in structural index. For one-time-pad encryption, the keys are the same length as that of the plaintext values. Furthermore, every key is used only once. Therefore, assume there are $n$ distinct tags in structural index, $|Key II| = \sum_{i=1}^{n} |t_i|$, where $t_i$ is a plaintext tag that is encrypted in structural index. It is straightforward that the upperbound of $|Key II|$ is $n \times max_i(|t_i|)$, which is proportional to the size of the original database.

Size of Key III: The size of key III is decided by splitting. The number of keys is bounded by $F/f$, where $F$ and $f$ are the maximum and minimum of number of frequencies of data values in the original database. Then $|Key III| = \frac{F}{f} \times l$, where $l$ is the length of key and is lower bounded by $\log(F/f)$. It is possible that frequency distribution of the original database is so skewed that $F/f$ is close to the number of tuples in the database (e.g., for the database of 1000 tuples containing a value of 990 number of occurrences, and 10 other values, each of number of occurrence 1, $F/f = 990$). To reduce the expensive space cost for this
case, we can first apply scaling (the same as our "scaling" in OPESS scheme) on the values of minimum number of frequencies. Let $B$ be the scaling factor. Then $|Key III|$ changes to $\frac{B^L}{B^L} \times l$ after scaling. Note that scaling introduces fake information of which encryption blocks contain which data values. Such fake information will increase the number of encryption blocks returned to the client and consequently add the post-processing work on the client side. Therefore, regarding scaling, there exists trade-off between number of keys that will be stored on the client side and the performance of client's query post-processing.

### 4.6.2 Client's Benefit

By the time happened, the client's activities can be categorized into three stages: setup stage, query stage, and post-process stage. Next, we discuss more details of the client's overhead cost for each stage.

During setup stage, the client has to deal with encryption, key management and metadata construction. Much of the work could be prepackaged and automated. For example, the client can use some software (e.g. Vernafi software [114]) to automate the key management. These processes can reduce the amount of the work for the client to setup the system to be minimal.

During query stage, the client writes queries and translates them before he/she sends them to the server. First, regarding query writing, he/she can use some graphical interface GUI (e.g. [109]) to help him/her to raise queries. Since his/her queries are simple XPath queries and we assume they have some background knowledge of how to write these queries, the cost of query writing is negligible. Second, he/she can make use of query rewriting program (which could be provided by the server) and key management software (again) for query translation. Our experimental result demonstrated that even for large queries, the query translation time is negligible compared with query processing time on the server. More details can be found in Section 4.7.4.

During post-process stage, the client decrypts the returned blocks and post-process the queries over the decryption result. Compared with query processing on the server side, these procedures are lightweight. Our experimental result demonstrate it. Section 4.7.3 has more details.

To summarize, although the client has a few activities to carry out, he/she is still benefited, since compared with the overhead cost of these activities, he/she gains more saving in data management, especially in query processing and data maintenance.

### 4.7 Experimental Evaluation

We ran a battery of experiments to measure the query evaluation performance on both client and server sides, and explored various factors that impact query performance, e.g., various encryption schemes, query size, etc. In this section, we describe our experiments and provide an analysis of our observations.
4.7.1 Experimental Setup

Setup: We used one SPARC workstation running SunOS 5.9 with 8 processors each having a speed of 900 MHz and 32 GB of RAM as a server, and one workstation running Linux version 2.6.5 with 1 processor having a speed of 900 MHz and 512 MB of RAM as a client. The communication speed between these two machines was 100 Mbps.

Data sets: We used both synthetic and real data sets for the experiments. The synthetic data we used is from the XMark benchmark. The real dataset is the NASA database from the XML data repository of the University of Washington.

Security Constraints: We defined a set of security constraints on both the XMark database and the NASA database, which are represented by the constraint graph (which has a node for every tag appearing in the security constraints and an edge representing every association type SC).

Encryption Schemes: To study the influence of different encryption schemes on the query-processing performance, we designed the following four schemes for our experiments:

1. Optimal (or opt) scheme: the document is encrypted by the optimal secure encryption scheme. The optimal encryption scheme encrypts the nodes name and creditcard on the XMark database, and encrypts the nodes initial and last on the NASA database.

2. Approximate (or app) scheme: the document is encrypted by the approximate scheme obtained by Clarkson’s approximation algorithm. The approximation encryption scheme encrypts nodes name, income and address on XMark database, and encrypts nodes initial, title, date, publisher, age, and city on NASA database.

3. Sub scheme: the document is encrypted on those nodes that are parents of the encrypted node in Scheme 1.

4. Top scheme: the whole document is encrypted as one block.

Query Set: We created three kinds of queries for each encrypted document: (1) \( Q_s \), the queries output the children node of the root of the document, (2)
Chapter 4. Query Evaluation over Encryption XML Databases

$Q_m$, the queries output the nodes on the $[h/2]$ level, where $h$ is the depth of the document tree, and (3) $Q_l$, the queries output the leaf nodes. For each category of queries, we created 10 queries and reported the average.

We used the XQEngine engine[113] for query evaluation. All values reported are the average of five trials after dropping the maximum and minimum for different workloads.

### 4.7.2 Our Approach vs. Naive Method

![Figure 4.12: Saving Ratio by Our Approach, XMark, 10MB](image)

Figure 4.12: Saving Ratio by Our Approach, XMark, 10MB

The naive method is that the server sends the whole encrypted document to the client, the client decrypts it and applies the query on the decrypted result. We compare the performance of our techniques with the naive method and measure the savings by our approach. The saving ratio $S_Q$ is defined as $S_Q = \frac{T_n - T_a}{T_n}$, where $T_n$ is the time taken to evaluate a query $Q$ by the naive method, and $T_a$ is the time taken by our technique. Intuitively, the closer the saving ratio to 1, the more time is saved. We ran the experiments for the databases encrypted by various encryption schemes and observed that our method achieves a substantial saving ratio: the ratio varies from 0.72 to 0.89 for both the XMark database and the NASA database. Furthermore, the finer grained the encryption scheme is, the larger the saving ratio. This is because the dominant performance factors, the query evaluation time on the server and the decryption time on the client, improves more significantly for finer encryption schemes. The results for the XMark database are shown in Figure 4.12.

### 4.7.3 Division of Work between Client and Server

We measured the following parameters for each query: query translation time on the client, query translation time on the server, query processing time on the server, transmission time of the answer from the server to the client, decryption time on the client, and query post-processing time on the client. Since we ran the experiment on a high-speed network, the transmission time is negligible compared with the other time factors. The query translation time on the client and the server are also both negligible: even for a document size of $50MB$ and
4.7.4 Effect of Query Size

We also measured the query-translation time on both the client and the server for queries of different sizes. The results showed that the query-translation time is negligible compared with the query-processing time: even for a document size of 50MB and a query of 20 nodes, the translation time on the client is less than 5 milliseconds and the query translation time on the server is only 13 milliseconds, which is only 1/3000 of the query-processing time on the server!

4.7.5 Effect of Various Encryption Schemes

First, we measured the query-evaluation performance for different encryption schemes. Figure 4.13 and Figure 4.14 show the results for the NASA and the XMark databases. The observation is that for the same query, the query-processing time on the server, the decryption time on the client and the query-processing time on the client decreases in order for the top, sub, app, and opt schemes. However, the query-processing time on the server decreases slower than the decryption time and query-processing time on the client, i.e., the improvement of query performance by the better encryption schemes is mainly on the client side. Another observation is that for all results, the query performance of the app scheme is around 1.1 - 1.3 times that of the opt scheme, i.e., the app scheme is a reasonable alternative to the opt scheme.

Second, we measured how the encryption time and the size of the encrypted document varies for various schemes. The results of encryption time and the ratio of the size of the encrypted document over the size of the unencrypted database are shown in Figure 4.15(a) and Figure 4.15(b). Figure 4.15(a) shows that the encryption time for the app scheme is the worst. This is because the number of elements that are encrypted by the app scheme is the largest out of the four schemes. Figure 4.15(b) shows that the document encrypted by the sub scheme is of the largest size. The sub scheme is worse than the top scheme because there are thousands of elements that will be encrypted by the sub scheme, each with extra encryption information (e.g., the nodes EncryptedType,
EncryptionMethod defined by W3C's XML encryption standard) added, while there is only one such element encrypted by the top scheme. The sub scheme is worse than the opt and the app schemes because each node encrypted by the opt and the app schemes is the child node of the nodes that are encrypted by the sub scheme. Also, the number of nodes encrypted by the app and the opt schemes is close to that encrypted by the sub scheme. Thus, the sub scheme always encrypts more data than the opt and the app schemes. The other thing to note is that the opt scheme always achieves the best performance in both encryption time and size. Similar results were observed from the XMark database (Figure 4.15(b)).

We also measured by how much the app scheme and the opt scheme outperform the top and the sub schemes, respectively. We use saving ratio to show this result. The saving ratio $S_{a/t}$ ($S_{a/s}$) of app over top (sub) is defined as $S_{a/t} = \frac{T_{t} - T_{a}}{T_{t}}$ ($S_{a/s} = \frac{T_{s} - T_{a}}{T_{s}}$), where $T_{t}$ is the query evaluation time by the top scheme, $T_{a}$ is the time by the app scheme, and $T_{s}$ is the time by the sub scheme. We also define $S_{o/1}$ and $S_{o/s}$ in a similar way. Intuitively, the closer the saving ratio is to 1, the better. The results are shown in Figure 4.16. The figure shows that both app and the opt schemes get better saving ratios over the top scheme than the sub scheme. It also shows that the saving ratio grows when the output node of query is closer to the leaf (e.g. the opt scheme reaches the highest ratio 0.64 over the top scheme and 0.53 over the sub scheme for query $Q_{1}$ on the NASA database). This is because when the output node is closer to the leaf, it is closer to the node that is encrypted by the opt and the app scheme. Consequently, the cost of decryption and query processing on the client is lower.

### 4.8 Summary

In this chapter, we considered the problem of efficient evaluation of XPath queries on (partially) encrypted XML databases hosted by a untrusted server. We proposed security constraints as a means for the client to specify her security requirements. We formally defined the attack model and semantics of security of encryption scheme, metadata and query processing. We showed our approach of constructing a secure encryption scheme and showed that finding an optimal secure encryption scheme is NP-hard. We proposed a metadata mechanism (including structural and value indices) at the server side that enables efficient query evaluation. We showed that with presence of the structural and value metadata, for a large set of candidate databases that do not contain any sensitive information, the attacker cannot distinguish them from the original hosted data. We also proved that even after observing a series of queries and responses, an attacker's belief probabilities about sensitive structural or value association information about the database is always no larger than a given security threshold. We complemented our analytical results with a comprehensive set of experiments. Our experiments show that our techniques can deliver the efficient query evaluation while ensure the enforcement of the security constraints.
Figure 4.13: Query Performance of Various Encryption Schemes, NASA Database, 25MB
Figure 4.14: Query Performance of Various Encryption Schemes, XMark Database, 10MB

Figure 4.15: Encryption Times and Sizes regarding Various Encryption Schemes
Chapter 4. Query Evaluation over Encryption XML Databases

Figure 4.16: App and Opt Encryption Schemes vs Top and Sub Encryption Schemes
Chapter 5

Probabilistic Analysis of Privacy Leakage of Published Views

In the second part of this thesis, we study the problem of publishing relational databases for both privacy and utility purposes. Many organizations are increasingly publishing data that contains information of individuals. This information can be analyzed to find the hidden properties of a population. Study of the characteristics of population is vital to society. However, since the published data contains personal information, protecting an individual’s privacy is also important. To protect such privacy information, the data owner has to process the data before publishing it. We assume that both privacy and utility take the form of associations, which are pairs of data values in the same tuple.

K-anonymity [79] is a popular technique for privacy-preserving data publishing. The basic idea is to apply generalization to the data so that the data values in the table are replaced with less specific information (e.g., “city” will be replaced with “country”). After such generalization, there are at least $k$ tuples that are of the same quasi identifiers and cannot be distinguished from each other. For example, Figure 5.1 (b) is a 3-anonymity table, in which the first (and the last) three tuples have the same quasi identifier values (on the attributes Age, Sex, and City) and are indistinguishable from each other. However, although the privacy of data is protected by generalization, since the data is modified, the published data may become unusable for data analysis purposes. For example, from the released 3-anonymity table in Figure 5.1 (b), the analysis of in which districts people are more likely to have diarrhea, or at which ages people are more likely to have a heart attack, cannot return useful answers anymore. It is even worse that in order to guarantee the privacy, the data may have to be anonymized with unacceptable amount of information loss[3]. Thus instead of data generalization by k-anonymity, in this chapter, we consider an alternative approach: publishing data by using views. The basic idea is that the base table is decomposed into several view tables. No data value in the published views will be altered. Thus, if we put every pair of two attributes whose association must be preserved in the same view table, by using the published views, the same data analysis result is still obtainable.
Chapter 5. Probabilistic Analysis of Privacy Leakage of Published Views

<table>
<thead>
<tr>
<th>Name</th>
<th>Age</th>
<th>Sex</th>
<th>City</th>
<th>Disease</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sarah</td>
<td>30</td>
<td>F</td>
<td>Vancouver</td>
<td>Diarrhea</td>
</tr>
<tr>
<td>Betty</td>
<td>45</td>
<td>F</td>
<td>Vancouver</td>
<td>Heart Attack</td>
</tr>
<tr>
<td>Alice</td>
<td>30</td>
<td>F</td>
<td>Toronto</td>
<td>AIDS</td>
</tr>
<tr>
<td>John</td>
<td>30</td>
<td>M</td>
<td>Vancouver</td>
<td>Diarrhea</td>
</tr>
<tr>
<td>Alan</td>
<td>45</td>
<td>M</td>
<td>Vancouver</td>
<td>Heart Attack</td>
</tr>
<tr>
<td>Tim</td>
<td>30</td>
<td>M</td>
<td>Alberta</td>
<td>Diarrhea</td>
</tr>
</tbody>
</table>

(a): Base Table $T$

<table>
<thead>
<tr>
<th>Age</th>
<th>Sex</th>
<th>City</th>
<th>Disease</th>
</tr>
</thead>
<tbody>
<tr>
<td>[30, 45]</td>
<td>F</td>
<td>Canada</td>
<td>Diarrhea</td>
</tr>
<tr>
<td>[30, 45]</td>
<td>F</td>
<td>Canada</td>
<td>Heart Attack</td>
</tr>
<tr>
<td>[30, 45]</td>
<td>M</td>
<td>Canada</td>
<td>AIDS</td>
</tr>
<tr>
<td>[30, 45]</td>
<td>M</td>
<td>Canada</td>
<td>Heart Attack</td>
</tr>
<tr>
<td>[30, 45]</td>
<td>M</td>
<td>Canada</td>
<td>Diarrhea</td>
</tr>
</tbody>
</table>

(b): A 3-anonymity table

Figure 5.1: Example of K-anonymity

<table>
<thead>
<tr>
<th>Name</th>
<th>Sex</th>
<th>City</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sara</td>
<td>F</td>
<td>Vancouver</td>
</tr>
<tr>
<td>Betty</td>
<td>F</td>
<td>Vancouver</td>
</tr>
<tr>
<td>Alice</td>
<td>F</td>
<td>Toronto</td>
</tr>
<tr>
<td>John</td>
<td>M</td>
<td>Vancouver</td>
</tr>
<tr>
<td>Alan</td>
<td>M</td>
<td>Vancouver</td>
</tr>
<tr>
<td>Tim</td>
<td>M</td>
<td>Alberta</td>
</tr>
</tbody>
</table>

$\nu_1 = \Pi_{Name, Sex, City}(T)$

$\nu_2 = \Pi_{Age, City, Disease}(T)$

Figure 5.2: Example of Unsafe Views

<table>
<thead>
<tr>
<th>Name</th>
<th>Age</th>
<th>Sex</th>
<th>City</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sarah</td>
<td>30</td>
<td>F</td>
<td>Vancouver</td>
</tr>
<tr>
<td>Betty</td>
<td>45</td>
<td>F</td>
<td>Vancouver</td>
</tr>
<tr>
<td>Alice</td>
<td>30</td>
<td>F</td>
<td>Toronto</td>
</tr>
<tr>
<td>John</td>
<td>30</td>
<td>M</td>
<td>Vancouver</td>
</tr>
<tr>
<td>Alan</td>
<td>45</td>
<td>M</td>
<td>Vancouver</td>
</tr>
<tr>
<td>Tim</td>
<td>30</td>
<td>M</td>
<td>Alberta</td>
</tr>
</tbody>
</table>

$\nu_1 = \Pi_{Name, Age, Sex, City}(T)$

$\nu_2 = \Pi_{Age, Disease}(T)$

Figure 5.3: Example of Safe Views
of in which districts people are likely to have diarrhea is exactly the same as
determined from the base table. The analysis of at which ages people are likely
to suffer a heart attack shows a similar result.

Publishing data by views solves the utility problem. It may initially seem
that the protection of private association is also straightforward: we put any
pair of two attributes whose association must be protected into different view
tables. However, as pointed out by [94], published views may still explicitly
reveal private associations; Figure 5.2 shows such an example. By joining \( v_1 \)
and \( v_2 \), which are the published result of the base table in Figure 5.1 (a), \( Alice \)
will be associated with only AIDS. Thus, the attacker can infer the sensitive
information that \( Alice \) has AIDS with 100% certainty. Figure 5.3 shows another
possible published view, by which \( Alice \) will be associated with both AIDS and
diarrhea in the join result of view tables. The attacker cannot explicitly decide
that \( Alice \) has AIDS. In this case, his/her belief is definitely less than 100%.

Since different schemes yield different probabilities of privacy breach, a for­
mal definition and thorough study of the privacy breach of the published views
is an important problem, and is the main focus of this chapter. Specifically,
based on our notions of possible worlds and interesting worlds, we propose a
generic framework to measure privacy breach of published views. We propose
two different attack models and for each model, we analyze the probability of
privacy breach. We observe that the probability of security breach of our re­
stricted attack model is related to the \( k \)-anonymity measurement developed in
[94].

**Contributions** Our contributions in this chapter include the following:

- We propose the secret query notion to specify private associations of a
given database (Section 5.2.1). The secret query can be expressed in SQL.
- We define two attack models, the unrestricted and restricted models, and
define the probability of a privacy breach of each model (Section 5.2.2).
- We propose a novel structure, the connectivity graph, as the synopsis of
the base table (Section 5.3).
- By using the connectivity graph, we derive the formulas of quantifying the
probabilities of privacy breach for each attack model when there are two
view tables to join. (Section 5.4.1).
- We discuss the difficulty of measuring the probability of privacy breach
when there are more than two view tables available (Section 5.4.2).

We use the notations shown in Figure 5.4 in this chapter and the next chap­
ter. We discuss the related work of this chapter in Section 5.1 and summarize
this chapter in Section 5.5. The work was published on the workshop on Privacy
in the Electronic Society (WPES) 2006 [90].
5.1 Related Work

Various aspects of the problem of protecting privacy by published views have been studied. The work by Chao et al. [94] is the most closely related to ours. They studied the problem of whether the released views violate the $k$-anonymity constraint for a set of secret associations. They used association cover, which is the collection of the property values whose corresponding ID values are the same, to measure information disclosure. By their definition, the released views violate the $k$-anonymity requirement if there are association covers of size less than $k$. However, we consider a different measurement, probabilistic analysis, to gauge information leakage. We define two attack models, the unrestricted and the restricted model, and measure the privacy breach of each model. We show that a connection exists between [94] and our work: the probability of breach of our restricted model is equivalent to $1/n$, whereas $n$ is the size of association cover in [94].

Miklau et al. [72] studied the problem of whether publishing view $V$ logically discloses any information about a confidential query $Q$. Deutsch et al. [38] re-examined the problem by taking the additional correlations between tuples into consideration. For both [72] and [38], the published views $V$ are secure if the a posteriori probability of any possible answer to $Q$ equals to its a priori probability. Both [72] and [38] assume the attacker knows how to compute the probability. Thus, they mainly focus on the complexity result for deciding whether a query $Q$ is secure with regard to the published view $V$. However, we focus on the details of computation of the probabilities for different attack models. Furthermore, we focus on the protection of associations between data values, while they consider the protection of the answer of a particular secret query.

Recently, Malin et al. [67] proposed the concept of re-identification. Specifically, people's identification will be re-identified by trailing their information that they leave at multiple locations, although the information on every single location is anonymous. They studied the problem of constructing trails across
websites from small amounts of seemingly anonymous evidence that a person has visited there. Trails are also constructed on places where the person has left explicit information of their presence. By identifying uniqueness and inferences across these two sets of the trails, the person's identification and their trails will be related. Compared with their work, they focus on trails of people's identification across websites, while we consider across multiple view tables.

Nergiz et al. [75] extends k-anonymity definitions for the multi-relational setting, in which information about a private entity is contained in multiple tables, and not easily represented in a single table. They propose a new clustering algorithm to achieve multi-relational anonymity. Their approach is on data generalization/suppression, while we consider no data modification.

5.2 Security Model

In this section, we define private association and the various attack models.

5.2.1 Private Association

As in [94], we consider that the base table $T$ contains an identifier attribute $ID$ and a property attribute $P$, where $ID$ uniquely identifies an entity (e.g., the attribute Name in Figure 5.1), and $P$ is a private property of the entity (e.g., the attribute Disease in Figure 5.1). We specify the private association $\alpha$ as $(ID = i, P = p)$, which denotes the fact that the entity $i$ associated with property $p$ must be protected. Every private association $\alpha$ can be specified as a SQL query $q^\alpha$, with $ID$ and $P$ being put in the select clause and value-based constraints in the where clause. For example, the association $(Name=*, Disease=AIDS)$ can be specified as "select Name, Disease from T where Disease=AIDS". We assume that for any private association $(ID, P)$, each value on $ID$ is associated with at most one value on $P$ in the base table. We call this property the uniqueness property.

5.2.2 Attack Models

5.2.2.1 Unrestricted Attack Model

We first consider an attack model in which the attacker does not know that the uniqueness property holds for the base table. Then by accessing the published views $V$, the attacker can try to reason about the possible base tables that would yield the same tables as $V$ by using the same view definitions of $V$. Notice that hiding the view definitions from the attacker does not really help, so we should consider the case where the view definitions are known to the attacker. We formalize this idea next, and identify each possible database as an unrestricted possible world.

Definition 5.2.1 [Unrestricted Possible Worlds] Given a base table $T$ and a view $V=\{V_1, V_2, \ldots, V_n\}$, let $q_i$ be the definition of view $V_i$. Then the
unrestricted possible worlds of $T$, $UPW_T = \{T' \mid T' \text{ is a relation of the same schema as } T, \text{ and for every view definition } q_i, q_i(T') = q_i(T)\}$. 

To understand the definition better, let us look at an example. Suppose the base table $T$ shown in Figure 5.5(a) is decomposed into two views, $V_1 = \Pi_{A,B}(T)$ and $V_2 = \Pi_{B,C}(T)$ (Figure 5.5 (b) and (c)). The unrestricted possible worlds $UPW_T$ are shown in Figure 5.5 (d). By applying the same view definition $\Pi_{A,B}$ and $\Pi_{B,C}$ on each possible world in Figure 5.5 (d), the same view tables as $V_1$ and $V_2$ are yielded.

Out of all unrestricted possible worlds, only a subset of those worlds contains the private association. This subset of possible worlds can help the attacker infer the existence of a private association in the base table. We call such databases unrestricted interesting worlds. The formal definition is as follows:

**Definition 5.2.2 [Unrestricted Interesting Worlds]** Given a table $T$ and an association $\alpha$, whose corresponding query is $q^\alpha$, the unrestricted interesting worlds of $\alpha$ with regard to $T$ $UIW^T_T = \{T' \mid T' \in UPW_T, \text{ and } q^\alpha(T') \neq \emptyset\}$. 

As an example, from the seven unrestricted possible worlds in Figure 5.5 (d), the unrestricted interesting worlds of the association $\alpha(A = a_1, C = c_1)$ are $\{w_1, w_3, w_4, w_5, w_7\}$.

### 5.2.2.2 Restricted Attack Model

It is possible that the attacker may know that the uniqueness property holds for the base table. Thus, he will only consider the database candidates that
Figure 5.6: Probability of Privacy Breach of Two Attack Models

satisfy the uniqueness property. To be more precise, we provide the following definitions.

**Definition 5.2.3 [Restricted Possible Worlds]** Given a base table \( T \) and a view \( V = \{V_1, V_2, \ldots, V_n\} \), let \( \alpha \) be the private association. Then the restricted possible worlds of \( \alpha \) with regard to \( T \)
\[
RPW^\alpha = \{T' \mid T' \in UPW_T, \text{ and } |C^\alpha_T| = 1\}. \]

For the same example as above, the restricted possible worlds of the association \( \alpha(A = a_1, C = c_1) \) are \( \{w_1, w_2, w_4, w_6\} \), where in each world \( a_1 \) is associated with either \( c_1 \) or \( c_2 \), but not both of them at the same time. Note that the association \( (a_1, c_2) \) does not exist in the base table. However, it exists in \( RPW^\alpha_T \).

The attacker will apply the same reasoning on the interesting worlds. Formally,

**Definition 5.2.4 [Restricted Interesting World]** Given a table \( T \), a view \( V \), and an association \( \alpha \), whose corresponding query is \( q^\alpha \). The restricted interesting worlds of \( \alpha \) with regard to \( T \)
\[
RIW^\alpha = \{T' \mid T' \in RPW^\alpha_T, \text{ and } q^\alpha(T') \neq \emptyset\}. \]

Still using the same example, from the four restricted possible worlds \( \{w_1, w_2, w_4, w_6\} \), \( RIW^\alpha_T = \{w_1, w_4\} \).

### 5.2.2.3 Probability of Privacy Breach

We assume that every possible world and interesting world are equally likely. Then for a given association \( \alpha \) in both attack models, we define the probability of privacy breach of \( \alpha \) based on unrestricted and restricted possible and interesting worlds. Figure 5.6 shows the details of this approach.

For the running example of association \( \alpha(A = a_1, C = c_1) \), the probability of privacy breach of the unrestricted model is \( 5/7 \), which means the attacker’s belief that the association \( (A = a_1, C = c_1) \) exists in the base table is \( 5/7 \). For the restricted model, the probability is \( 2/4 = 1/2 \).

### 5.3 Connectivity Graph

The naive method of measuring the probability of privacy breach is to reconstruct both possible worlds and interesting worlds (see the previous section for their definitions) and count their numbers. However, since the numbers of both worlds are exponential in the number of tuples in the views, the reconstruction cost will be very expensive. Thus, we use a connectivity graph to avoid the reconstruction and help to efficiently calculate the probability of privacy
Figure 5.7: Connectivity Graph of the Two View Tables in Figure 5.3

breach. In this section, we explain the details of the connectivity graph for both two-view-table and k-view-table (k>2) cases.

5.3.1 Two-View-Table Case

Given a view $V$ that consists of two view tables, the connectivity graph is the following steps:

**Step 1: Locate Relevant Tuples**

For a given base table $T$ and an association $\alpha$, we first locate the tuples in $T$ that contain $\alpha$ by applying $q^\alpha$ on $T$, where $q^\alpha$ is the corresponding query of the association $\alpha$. Then we locate the tuples $t \in T$ such that there exists $t' \in q^\alpha(T)$ such that $t$ and $t'$ have the same values on the join attributes of the views. The tuples returned by the above procedure are considered to be relevant tuples. For example, for the base table $T$ in Figure 5.1 (a) and the association $\alpha(\text{Name}=\text{Alice}, \text{Disease}=\text{AIDS})$, $q^\alpha(T)=\{\text{tuple 3}\}$. Based on the two released views in Figure 5.3 whose join attribute is $\text{Age}$, since tuple 3 has the same value 30 of attribute $\text{Age}$ as that of tuples 1, 4 and 6, tuples 1, 3, 4 and 6 are returned as the relevant tuples of $\alpha$. After we locate the relevant tuples in the base table (tuple 1, 3, 4 and 6 in the example), we highlight the corresponding tuples in the view tables.

**Step 2: Construct the Connectivity Graph**

We construct the connectivity graph $CG$ as follows: for every view $V_i$, let $U_{V_i}$ be the set of the attributes of $V_i$ that are either join attributes or involved in the definition of sensitive association. For example, for the two views $V_1 = \Pi_{\text{Name},\text{Age},\text{Sex},\text{City}}(T)$ and $V_2 = \Pi_{\text{Age},\text{Disease}}(T)$ in Figure 5.3, $U_{V_1}=$\{Name, Age\} and $U_{V_2}=$\{Age, Disease\}. Then for every highlighted tuple $t \in V_i$, there is a node $n \in CG$ corresponding to $\Pi_{U_{V_i}}(t)$, i.e., the projection on attributes $U_{V_i}$ of tuple $t$. For every two nodes $n_1$ and $n_2$, there is an edge between $n_1$ and $n_2$ if they correspond to two tuples that have the same value on the join attributes. As an example, Figure 5.7 shows a connectivity graph for the two views $V_1 = \Pi_{\text{Name},\text{Age},\text{Sex},\text{City}}(T)$ and $V_2 = \Pi_{\text{Age},\text{Disease}}(T)$ in Figure 5.3.

Note that the connectivity graph is association-specific. Different associations result in different connectivity graphs. One nice property of the connectivity graph of two-view-table case is that it is always a bipartite complete graph (e.g., the connectivity graph $CG$ in Figure 5.7). We will show how to use this
property to calculate the probability of privacy breach in Section 5.4.1.

5.3.2 K-View-Table Case

If the view \( \mathcal{V} \) consists of \( k > 2 \) view tables, we can also construct the connectivity graph for this case. Compared with the two-view-table case, the difficulty of the k-view-table case is that more than one join attribute exist. For example, consider the base table in Figure 5.1 (a), and the four view tables \( V_1' = \Pi_{Name, Age}(T) \), \( V_2' = \Pi_{Age, Sex}(T) \), \( V_3' = \Pi_{Sex, City}(T) \) and \( V_4' = \Pi_{City, Disease}(T) \). For the view \( V_2' \), its join attribute with \( V_1' \) is \{Age\}, and with \( V_3' \) is \{Sex\}. Thus, if we only consider one join attribute to locate the relevant tuples, we may miss some possible associations constructed by the joins and consequently wrongly conclude the number of possible worlds and interesting worlds. Thus, we do the following modification for the construction of the connectivity graph: for the first step, "locate relevant tuples", we assume there are \( n \) possible join attributes in total. Then for each join attribute, we apply the same reasoning as for the two-view-table case and locate a set of relevant tuples. We union those relevant tuples together and put all of them into connectivity graph by the same Step 2 as for the two-view-table case. For example, consider the base table in Figure 5.1 (a), and the four view tables \( V_1' = \Pi_{Name, Age}(T) \), \( V_2' = \Pi_{Age, Sex}(T) \), \( V_3' = \Pi_{Sex, City}(T) \) and \( V_4' = \Pi_{City, Disease}(T) \). There are three join attributes: \( Age \) (by \( V_1' \) with \( V_2' \)), \( Sex \) (by \( V_2' \) with \( V_3' \)) and \( City \) (by \( V_3' \) and \( V_4' \)). Then for the association \( \{Name=\text{Alice}, Disease=\text{AIDS}\} \), we locate tuples 1, 3, 4 and 6 for attribute \( Age \). For attribute \( Sex \), we locate tuple 1, 2 and 3. And for attribute \( City \), we locate tuples 1, 2, 4 and 5. Figure 5.8 shows the corresponding connectivity graph. It consists of all relevant tuples (tuple 1, 2, 3, 4, 5 and 6) in the base table. We assume there is only one join path of the view tables.

As we have shown, any connectivity graph that is constructed for a two-view-table case is a complete bipartite graph. For k-view-table \((k > 2)\) case, although the connectivity graph is still a k-partite graph \((k \) is the number of views\), it is no longer a complete graph. In fact, the structure of the graph depends highly on the data in the base table. In Section 5.4.2, we will show why this property poses problems for the calculation of sizes for both possible worlds and interesting worlds.

5.4 Measurement of Probability of Privacy Breach

In this section, we discuss how to use the connectivity graph to calculate the probabilities of privacy breach of each attack model for both the two-view-table and the k-view-table \((k>2)\) cases.
5.4.1 Two-View-Table Case

We first define the *unrestricted cover* and *unrestricted interesting cover* of the connectivity graph.

**Definition 5.4.1 [Unrestricted Cover]** Let \( G = (N, E) \) be a bipartite graph, where \( N = N_1 \cup N_2 \). Let \( l \in N_1 \) and \( r \in N_2 \) be two distinguished nodes. An *unrestricted cover* of \( G \) is a subset of edges \( C \subseteq E \) such that for every node \( n \in N \), there exists an edge \( e \in C \) such that \( e \) is incident on \( n \). Furthermore, an unrestricted cover \( C \) is said to be *interesting* provided \( C \) contains at least one path from \( l \) to \( r \).

Intuitively, the *unrestricted cover* is an edge set such that every node is covered by at least one edge in it. For example, one unrestricted cover of the connectivity graph of Figure 5.7 is \{< 1,6 >, < 2,5 >, < 2,6 >, < 3,5 >, < 4,6 >\}. If we specify \( l \) as node 1 and \( r \) as node 6, this cover is an unrestricted interesting cover.

An unrestricted (interesting) cover is related to an unrestricted (possible or interesting) world. We have the following lemma:

**Lemma 5.4.1 (Unrestricted Covers vs. Unrestricted Worlds):** Given a base table \( T \) and a view \( V \), let \( CG \) be the corresponding connectivity graph of \( V \). For any association \( \alpha(ID = i, P = p) \), let \( n_i \) and \( n_j \) be the nodes in \( CG \) that correspond to the tuples that contain \( ID = i \) and \( P = p \). Then the number of unrestricted possible worlds equals the number of *unrestricted covers* of \( CG \). Furthermore, the number of unrestricted interesting worlds equals the number of *unrestricted interesting covers* of \( CG \) by treating \( n_i \) and \( n_j \) as the distinguished nodes \( l \) and \( r \).

The correctness of Lemma 5.4.1 is based on the definition of unrestricted worlds and unrestricted covers. For example, for the bipartite graph in Figure 5.7, Figure 5.9 shows a subset of unrestricted covers and the corresponding possible worlds. For the association \( \alpha(\text{Name}=\text{Alice}, \text{Disease}=\text{AIDS}) \), whose corresponding nodes in Figure 5.7 are node 2 and 6, its unrestricted interesting worlds are the subset of unrestricted covers that contain the edge \(< 2,6 >\).

In general, we have the following lemma:
Chapter 5. Probabilistic Analysis of Privacy Leakage of Published Views

<table>
<thead>
<tr>
<th>ID</th>
<th>Unrestricted Cover</th>
<th>Possible World</th>
</tr>
</thead>
<tbody>
<tr>
<td>$w_1$</td>
<td>$&lt;1,5&gt;$, $&lt;2,6&gt;$, $&lt;3,5&gt;$, $&lt;4,5&gt;$</td>
<td>{Sarah, 30, Diarrhea}, {Alice, 30, AIDS} {John, 30, Diarrhea} {Tim, 30, Diarrhea}</td>
</tr>
<tr>
<td>$w_2$</td>
<td>$&lt;1,6&gt;$, $&lt;2,5&gt;$, $&lt;3,6&gt;$, $&lt;4,6&gt;$</td>
<td>{Sarah, 30, AIDS}, {Alice, 30, Diarrhea} {John, 30, AIDS} {Tim, 30, AIDS}</td>
</tr>
</tbody>
</table>

...  

Figure 5.9: Covers vs. Possible Worlds

Lemma 5.4.2 (Number of Unrestricted Possible & Interesting Worlds): For a bipartite graph with two partitions consisting of $m$ and $n$ nodes, respectively, the number of unrestricted possible worlds

$$UPW(m, n) = \sum_{i=0}^{m} \sum_{j=0}^{n} (-1)^i (-1)^j \binom{m}{i} \binom{n}{j} 2^{(m-i)(n-j)}.$$ 

The number of unrestricted interesting worlds

$$UIW(m, n) = \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} (-1)^i (-1)^j \binom{m-1}{i} \binom{n-1}{j} 2^{(m-i)(n-j)-1}.$$ 

Furthermore, when either $m$ or $n$ equals 1, $UPW(m, n) = UIW(m, n) = 1$.

Proof:

Given a bipartite graph $G$ with two partitions consisting of $m$ and $n$ nodes (we call them $m$-partition and $n$-partition), we first count the total number of graphs that cover the nodes in the $m$-partition. Each node $v$ in the $m$-partition has $n$ potential edges connected with the nodes in the $n$-partition. Every subset of these $n$ edges covers node $v$ except for the empty subset. Therefore, node $v$ has $2^n - 1$ possible set of edges, any one of which guarantees that $v$ is covered. Moreover, each node in the $m$-partition can pick its subset of edges independently. Hence, the number of graphs that cover the nodes in the $m$-partition is

$$(2^n - 1)^m \quad (5.1)$$

Second, we calculate the number of graphs in which there exists a node in the $n$-partition that is not covered and eliminate this number from Eq 5.1. Let $P$ be all the graphs represented by Eq 5.1. Let $A_i$ be a subset of $P$ in which there exists a node $i$ in the $n$-partition that is not covered. It is straightforward that $| (P - (A_1 \cup A_2 \cup ... \cup A_n) |$ (i.e., remove $(A_1 \cup A_2 \cup ... \cup A_n) $ from $P$) is the number of the covers we are looking for (i.e., the covers in which every node in the $m$- and $n$-partition has been covered). Since $| (P - (A_1 \cup A_2 \cup ... \cup A_n) |$ equals $| P | - | (A_1 \cup A_2 \cup ... \cup A_n) |$, and $| P | = (2^n - 1)^m$, we next calculate $| A_1 \cup A_2 \cup ... \cup A_n |$.

By inclusion/exclusion, $| A_1 \cup A_2 \cup ... \cup A_n |$ can be converted to $\Sigma_i | A_i | - \Sigma_{i,j,i\neq j} | A_i \cap A_j | + \Sigma_{i,j,k,i\neq j,k} | A_i \cap A_j \cap A_k | - ...$. Specifically, $A_i$ stands
Chapter 5. Probabilistic Analysis of Privacy Leakage of Published Views

for the edge set in which each node in the \( m \)-partition has no edge connected
with node \( i \) in the \( n \)-partition but has at least one edge with any other node
in the \( n \)-partition. Thus each node in \( m \)-partition has \( 2^{n-1} - 1 \) choices of edge
sets. Therefore, \(| A_i | = \binom{m}{i}(2^{n-1} - 1)^m \). Similarly, \(| A_i \cap A_j |\) stands for
the edge set in that each node in the \( m \)-partition has no edge connected to nodes
\( i \) and \( j \) in the \( n \)-partition. Thus, each node has \( 2^{n-2} - 1 \) choices of possible
edges. Hence \(| A_i \cap A_j | = \binom{m}{2}(2^{n-2} - 1)^m \). We can apply the similar reasoning
for \( \Sigma_{i,j,k,...,i\neq j\neq k...} | A_i \cap A_j \cap A_k ... | \). By putting them together we have:

\[
| A_1 \cap A_2 \cap ... \cap A_n | = \binom{m}{i}(2^{n-1} - 1)^m - \binom{m}{2}(2^{n-2} - 1)^m + \binom{m}{3}(2^{n-3} - 1)^m - .... \tag{5.2}
\]

Thus we have

\[
| P - (A_1 \cup A_2 \cup ... \cup A_n) | = \Sigma_{i=0}^{m} (-1)^i \binom{m}{i} (2^{n-i} - 1)^{m-1} \tag{5.3}
\]

By expanding \((2^{n-i} - 1)^{m-1}\) to be \( \Sigma_{j=0}^{m} (-1)^j \binom{m}{j} 2^{(n-i)(m-j)} \), we can rewrite Eq
5.3 as \( \Sigma_{i=0}^{m} \Sigma_{j=0}^{m} (-1)^i \binom{m}{i} \binom{m}{j} 2^{(n-i)(m-j)} \), which is the result of \( UPW(m, n) \).

We can derive the number of \( UIW(m, n) \) by the similar reasoning as above.

The difference between \( UIW \) and \( UPW \) is that the edge between a specific node
in the \( m \)-partition and a specific node in the \( n \)-partition must exist in the cover.
Without losing generality, we assume the first nodes in both the \( m \)-partition
and the \( n \)-partition are two such specific nodes. Therefore, when we reason
about the number of graphs that cover the nodes in the \( m \)-partition, since the
first node in the \( m \)-partition has \( n - 1 \) choices of edges, while the other \( m - 1 \)
nodes in the \( m \)-partition has \( n \) choices, it is

\[
| P | = 2^{n-1}(2^n - 1)^{m-1} \tag{5.4}
\]

We define \( A_i \cap A_j \ldots \) as the same as above. Then \(| A_i | = 2^{n-2} \binom{n-1}{1}(2^{n-1} - 1)^{m-1} \) (since the first node in the \( m \)-partition has \( n - 2 \) choices of nodes that
it has no edge connected with, while the other \( m - 1 \) nodes in the \( m \)-partition
has \( n - 1 \) choices of nodes that they have no edge connected with). Similarly
\(| A_i \cap A_j | = 2^{n-3} \binom{n-2}{1}(2^{n-2} - 1)^{m-1} \). We can apply similar reasoning for
\( \Sigma_{i,j,k,...,i\neq j\neq k...} | A_i \cap A_j \cap A_k ... | \). By putting them together we have:

\[
| A_1 \cap A_2 \cap ... \cap A_n | = 2^{n-2} \binom{n-1}{1}(2^{n-1} - 1)^{m-1} - 2^{n-3} \binom{n-2}{1}(2^{n-2} - 1)^{m-1} + ... \tag{5.5}
\]

Hence we have

\[
| P - (A_1 \cup A_2 \cup ... \cup A_n) | = \Sigma_{i=0}^{m-1} (-1)^i 2^{n-1-i} \binom{n-1}{i} (2^{n-i} - 1)^{m-1}. \tag{5.6}
\]

By expanding \((2^{n-i} - 1)^{m-1}\) to be \( \Sigma_{j=0}^{m-1} (-1)^j \binom{m-1}{j} 2^{(n-i)(m-1-j)} \), we can rewrite
Eq 5.6 as \( \Sigma_{i=0}^{m-1} \Sigma_{j=0}^{m-1} (-1)^i \binom{m-1}{i} \binom{m-1}{j} 2^{(m-i)(n-j)-1} \), which is the result
of \( UIW(m, n) \).
For the running example, the bipartite graph is partitioned into two disjoint sets, one of four nodes and one of two nodes. Thus \( UPW(4, 2) = 79 \) and \( UIV(4, 2) = 53 \).

Then we discuss the restricted attack model. We first define the restricted covers and interesting restricted covers of connectivity graphs.

**Definition 5.4.2 [Restricted Cover]** Let \( G = (N, E) \) be a bipartite graph, where \( N = N_1 \cup N_2 \). Let \( l \in N_1 \) and \( r \in N_2 \) be two distinguished nodes. A restricted cover of \( G \) is a subset of edges \( C \subseteq E \) such that for every node \( n \in N \), there is some edge \( e \in C \) such that \( e \) is incident on \( n \). Furthermore, there is only one node in \( N_2 \) that \( l \) connects to. A restricted covering \( C \) is said to be interesting if \( C \) contains the path from \( l \) to \( r \). 

Intuitively, the restricted cover is an edge set that covers every node by at least one edge in it. Furthermore, there is only one property node that the ID node \( l \) is connected to. For the running example, \( \{<1,5>, <2,5>, <3,6>, <4,6>\} \) is a restricted cover of the connectivity graph in Figure 5.7. If we specify \( l \) as node 3 and \( r \) as node 6, this cover is an interesting restricted cover.

Similarly we can relate the restricted covers to the restricted worlds. We have the following lemma:

**Lemma 5.4.3 (Restricted Covers vs. Restricted Worlds):** Given a base table \( T \) and a view \( V \), let \( CG \) be the corresponding connectivity graph of \( V \). For any association \( \alpha(ID = i, P = p) \), let \( n_i \) and \( n_j \) be the nodes in \( CG \) that correspond to the tuples that contain the attributes \( ID = i \) and \( P = p \). Then the number of restricted possible worlds equals the number of restricted covers of \( CG \). Furthermore, the number of restricted interesting world equals the number of restricted interesting covers of \( CG \) by treating \( n_i \) and \( n_j \) as two distinguished nodes \( l \) and \( r \).

The correctness of Lemma 5.4.3 is based on the definition of restricted worlds and restricted covers. We have:

**Lemma 5.4.4 (Number of Restricted Possible & Interesting Worlds):** For a bipartite graph with two partitions consisting of \( m \) and \( n \) nodes, respectively, the number of restricted interesting worlds \( RIW(m, n) = UPW(m-1, n) + UPW(m-1, n-1) \). The number of the restricted possible worlds \( RPW(m, n) = n \times (UPW(m-1, n) + UPW(m-1, n-1)) \). Furthermore, when both \( m \) and \( n \) equal 1, \( RIW(m, n) = RPW(m, n) = 1 \).

**Proof:**

The restricted interesting cover is a union of two edge sets (illustrated in Figure 5.10): (1) the edge set in which \( n_j \) is only connected with \( n_i \), with the edge set on the remaining nodes in \( N_1 \setminus \{n_i\} \) and \( N_2 \setminus \{n_j\} \) forming unrestricted covers, and (2) the edge set in which \( n_j \) is connected with other nodes besides \( n_i \), with the edge set on the remaining nodes in \( N_1 \setminus \{n_i\} \) and \( N_2 \) forming unrestricted covers. There is no overlapping between these two edge sets. Since the unrestricted cover in set (1) is of two partitions of \( m-1 \) and \( n-1 \) nodes, and the unrestricted cover in set (2) is of two partitions of \( m-1 \) and \( n \) nodes, the number of restricted interesting covers \( RIW(m, n) = UPW(m-1, n) + \)
Figure 5.10: Restricted Interesting Worlds vs. Unrestricted Possible Worlds

<table>
<thead>
<tr>
<th>Probability Model</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unrestricted model</td>
<td>$\frac{\text{RIW}(m,n)}{\text{UPW}(m,n)}$</td>
</tr>
<tr>
<td>Restricted model</td>
<td>$\frac{\text{UPW}(m-1,n-1) + \text{UPW}(m-1,n-2)}{n \times (\text{UPW}(m-1,n) + \text{UPW}(m-1,n-1))} = \frac{1}{n}$</td>
</tr>
</tbody>
</table>

Figure 5.11: Probability Models

$\text{UPW}(m-1,n-1)$, where $m$ and $n$ are the numbers of nodes in two partitions in the connectivity graph.

To derive the number of restricted possible worlds, we use the same reasoning as for restricted interesting worlds. The only difference is that, for the restricted interesting world, node $n_i$ can only connect with $n_j$, while in the restricted possible world, $n_i$ can connect with any node in the partition that $n_j$ belongs to. There are $n$ choices for the node that $n_i$ can connect with, where $n$ is the number of nodes in the partition that $n_j$ belongs to. Thus $\text{RPW}(m,n) = n \times \text{RIW}(m,n) = n \times (\text{UPW}(m-1,n) + \text{UPW}(m-1,n-1))$, where $m$ and $n$ are the numbers of nodes in two partitions in the connectivity graph.

Figure 5.11 shows the probability of privacy breach for each attack model. Note that for the restricted model, the probability of privacy breach is $1/n$, where $n$ equals the size of the association cover by the k-anonymity model in [94]. For example, for the view tables $V_1$ and $V_2$ shown in Figure 5.5, as we have shown in Section 5.2.2, the probability of the privacy breach of the restricted model equals $1/2$, while according to [94], since $a_1$ is associated with both $c_1$ and $c_2$ in the join result of $V_1$ and $V_2$, the cover of association $\alpha(a_1,c_1)$ is of size 2.

5.4.2 K-view-table Case

For the case where the base table is split into $k > 2$ view tables, the difference from the two-view-table case is that, unlike in the 2-view-table case where the connectivity graph is always a complete bipartite graph, the connectivity graph for the k-view-table case is no longer a complete graph (see Figure 5.8 for an example). In fact, the structure of the graph depends highly on the data in the base table. We cannot come up with a general formula to calculate the numbers of both possible worlds and interesting worlds. Therefore, we enumerate all
possible worlds and interesting worlds, and eliminate those that do not satisfy
the definition. The enumeration procedure is computationally expensive. We
leave the efficient procedure of calculating the probability of privacy breach for
k-view-table case as an open problem.

5.5 Summary

In this chapter, we studied how to formally measure the privacy breach of the
published views. Specifically we have done the following work: we defined a
general framework to measure the likelihood of privacy breach of the published
views by using probability. We proposed two attack models and for each attack
case; we derived the formulas to calculate the probability of privacy breach for
the 2-view-table case. We discussed the difficulty of measuring the probability
of privacy breach for the k-view table case ($k > 2$).
Chapter 6

Privacy and Utility Preserving Data Publishing by Views

In Chapter 5, we defined privacy issues for data publishing. In this chapter, we also take utility into consideration. We assume the utility takes form of correlations of data values. For example, from the medical database in Figure 6.1, the researchers want to study at which ages people are more likely to suffer a heart attack. The results, which show the characteristics of a population, is vital to society. We call these correlation patterns public associations.

As in Chapter 5, we assume that the private information takes the form of associations, which are pairs of data values in the same tuple (for example, the pair (Alice, AIDS) in Figure 6.1), in the input database. We call these associations private associations. In this chapter, we study the problem that given a set of public and private associations, how to come up with a view scheme that not only meets all public associations but also satisfies all private associations. Unlike in Chapter 5, we assume IDs are removed from the published data.

Contributions We make the following contributions in this chapter:

- We propose the notions of privacy and public constraints (Section 6.2.1). We adapt the definition of association cover in [94] to measure privacy (Section 6.2.2).

- To efficiently measure the size of association cover, we adapt the definition of connectivity graph in Chapter 5 and show how we can use it to calculate the size of association cover efficiently (Section 6.4).

<table>
<thead>
<tr>
<th>Name</th>
<th>Age</th>
<th>Sex</th>
<th>City</th>
<th>Disease</th>
<th>History</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sarah</td>
<td>30</td>
<td>F</td>
<td>Vancouver</td>
<td>Diarrhea</td>
<td>Cold</td>
</tr>
<tr>
<td>Betty</td>
<td>45</td>
<td>F</td>
<td>Vancouver</td>
<td>Heart Attack</td>
<td>Stroke</td>
</tr>
<tr>
<td>Alice</td>
<td>30</td>
<td>F</td>
<td>Toronto</td>
<td>AIDS</td>
<td>Cold</td>
</tr>
<tr>
<td>John</td>
<td>30</td>
<td>M</td>
<td>Vancouver</td>
<td>Diarrhea</td>
<td>Cold</td>
</tr>
<tr>
<td>Alan</td>
<td>45</td>
<td>M</td>
<td>Vancouver</td>
<td>Heart Attack</td>
<td>Stroke</td>
</tr>
<tr>
<td>Tim</td>
<td>30</td>
<td>M</td>
<td>Edmonton</td>
<td>Diarrhea</td>
<td>Cold</td>
</tr>
</tbody>
</table>

Figure 6.1: Base Table
• Given privacy and public constraints, there are exponentially many candidate view schemes that satisfy both. We motivate the problem of finding a view scheme that does not violate any privacy constraint and achieves maximum utility with minimum noise, called the \textit{optimal faithful and safe view scheme}. We show that finding the optimal faithful and safe view scheme is NP-hard (Section 6.5.1).

• For a given base table, we define the \textit{view lattice} for representing the candidate view schemes and the relationships between them (Section 6.5.1). We establish properties of the view lattice (Section 6.5.2), and use them to optimize the search procedure. We develop an efficient algorithm to find a locally optimal safe and faithful view scheme (Section 6.5.3).

• We complement our analytical result with a detailed set of experiments that demonstrate the effectiveness and efficiency of our approach (Section 6.6).

We discuss the related work of this chapter in Section 6.1 and summarize this chapter in Section 6.7.

6.1 Related Work

\textbf{Publishing by views:} Various aspects of the problem of protecting private information by published views have been studied. Miklau and Suciu [72] examine the problem of whether by publishing a view \( V \), \( V \) logically discloses any information about a secret query \( Q \). They present a novel information-theoretic standard similar to Shannon's definition of \textit{perfect secrecy} [82]. Following their standard, \( V \) is considered securely published if the a posteriori probability of any possible answer to \( Q \) is the same as its a priori probability. While [72] assumes that the tuples in the secret answer are independent events, Deutsch et al. [38] re-examine the problem of deciding the security guarantees when correlations between tuples are available as prior knowledge in the form of integrity constraints. They prove several complexity results on deciding privacy guarantees in the presence of integrity constraints. Chao et al. [94] study the view-publishing problem in relation to k-anonymity. By their definition, a view scheme \( V \) consisting of a set of tables violates k-anonymity if there exists a sensitive association of form \((ID = i, P = p)\) such that in the join result of \( V \), \( ID = i \) is associated with less than \( k \) values of the attribute \( P \), including \( p \). They show that when there is no functional dependency (i.e., the constraint between two sets of attributes) available to the attacker, the complexity of checking violation is polynomial time.

All of the papers mentioned above focus mainly on the \textit{verification} problem: given a set of view tables, whether they disclose any privacy information, while we study the \textit{design} problem: given the schema of a base table and a set of privacy and public constraints, how can we come up with a scheme for the published view that satisfies all constraints and achieves maximum utility? In
principle, we could use the procedure from [94] to check all possible candidate view schemes and determine the one(s) that satisfy all constraints. However, since the number of candidates is exponential, such procedure is prohibitively expensive. Thus, instead of the “generate and check” procedure, we propose a direct approach to designing view schemas that satisfy the specified constraints.

**K-anonymity:**  
*K-anonymity* is a popular approach to protecting private information in published data. There has been much research done on *k-anonymity* (e.g., [12], [69], [79], [65]). While most of the work focuses on generalization of data, which may ruin the utility of the data, we look at the problem of privacy-preserving data publishing from a different point of view: we split the base table into several view tables, without any modification to data values. Kifer et al. [54] study how to inject utility into generalized dataset. They propose publishing some anonymized marginals together with the generalized dataset so that those marginals can be used for data mining tasks. They extend the definitions of k-anonymity and l-diversity [65] from single-table anonymized data to collections of anonymized marginals. By their model, the closer the probability distribution of the anonymized data to that of the base table, the better utility it has. Wang et al. [92] propose a method for achieving k-anonymity across sequential releases of views. They study how to generalize the current table so that no private information will be released by joining the generalized version of the current table with a (set of) previously released table(s). They use entropy to measure the utility of the generalized data. Xiao et al. [87] present a novel technique, *anatomy*, for publishing the sensitive data without modifying the data in the base table. They combine the generalization technique with the view idea. By their approach, a quasi-identifier table, which contains the quasi-identifier attributes and the ID of the quasi-identifier group, and a sensitive table, which contains the ID of the quasi-identifier group and a sensitive attribute, will be published. They consider the result of aggregate queries alone, especially COUNT queries, as the measure of data utility. The above works adopt a different model from ours for privacy and utility. For privacy, they use associations, as with this work; they use different measures for utility.

### 6.2 Security Model

In this section, we first introduce the notion of private and public associations (Section 6.2.1). Then we formally define the measurement of privacy and utility (Section 6.2.2).

#### 6.2.1 Private and Public Associations

We assume that the base table $T$ contains an identifier attribute $ID$, forming a key of $T$, a set of *public* property attributes, which contains a set of quasi-identifier attributes $QID$ (i.e., a set of non-sensitive attributes that can be used to identify at least one individual from a given population by linking those attributes to external data sets), and a set of *sensitive* property attributes $S$. For instance, for the base table in Figure 6.1, the $ID$ attribute is *Name*, the
public property attributes are the same as $QID$ attributes, which are \{Age, Sex, City\}, and the $S$ attributes are \{Disease, History\}.

**Private associations:** We consider the associations between the ID attribute and sensitive property attributes as private. A private association $\alpha$ is specified as $\alpha = (ID=i, P=p) \ (P \in S)$, which declares that the fact that the entity $i$ is associated with sensitive property $p$ must be protected. For example, to specify the fact that Alice has AIDS must be protected, we would assert $\alpha = (Name=Alice, Disease=AIDS)$ as a private association. For any association $\alpha = (ID=i, P=p)$, we use $QID^\alpha$ to denote the (unique) set of values on $QID$ attributes that corresponds to $ID=i$, after $ID=i$ is removed from the base table. For example, for the association $\alpha = (Name=Alice, Disease=AIDS)$, $QID^\alpha = (Age=30, Sex=F, City=Toronto)$. Viewing $QID^\alpha$ as a predicate, by applying $\sigma_{QID^\alpha}$ ($\sigma$: the selection operator) to the base table, several tuples, including the one that satisfies $ID=i$, will be returned. For example, by applying $\sigma_{(Age=30, Sex=F, City=Toronto)}$ to the base table in Figure 6.1, tuple 3 will be returned, which contains Disease=AIDS.

**Public associations:** We assume that the data is published for the purpose of analysis of correlations of data values. For example, we might want to analyze which diseases are more likely to correlate with heart attack. We call such associations that must be preserved by published data public associations. Public associations are of the form $(P_1=p_1, P_2=p_2)$, which denotes that the fact that property $p_1$ and $p_2$ are associated together must be preserved in the published data. In this chapter, we only consider binary public associations.

With regard to private associations, protecting binary associations will clearly protect n-ary associations for $n > 2$. Previous literature mainly focuses on binary associations for privacy protection. For public associations, when an n-ary association needs to be preserved we can always capture it using $n(n-1)/2$ binary associations and ensure all the attributes involved are put together in one relation. This technique justifies our choice of binary associations as the focus of study for both public and private associations.

In the following discussion, we use the private and public association constraints specified in Example 6.2.1.

**Example 6.2.1 [Association Constraints]**

We specify the following association constraints for the base table in Figure 6.1:

**Private associations:**
1. Which person gets which disease: $(Name=*, Disease=*)$.
2. Which person once got which disease: $(Name=*, History=*)$.

**Public associations:**
1. For every specific disease, at which ages people are more likely to have it: $(Age=*, Disease=*)$.
2. For every specific disease, what are the past diseases that it is more likely to be correlated with: $(Disease=*, History=*)$.

**Consistency of Association Constraints**

Given a set of public associations $\Sigma_{pub}$, and a set of private associations $\Sigma_{pri}$, let $QID^\Sigma_{pri}$ be the set of QID
attributes of private associations in $\Sigma_{pub}$, and $\Sigma'_{pub}$ be the set of public associations such that $\forall \alpha(P_1 = p_1, P_2 = p_2) \in \Sigma_{pub}, (P_1, P_2) \in \Sigma'_{pub}$, and $\Sigma_{pri}$ are consistent if $QID_{pri} \cap \Sigma'_{pub} = \emptyset$. We can show that a given set of private (public) associations does not imply the privacy (preservation) of any other association. Thus, checking the consistency of the specification reduces to checking the disjointness of $QID_{pri}$ and $\Sigma'_{pub}$. This is a trivial check and in the following, we assume that the given sets $\Sigma_{pub}$ and $\Sigma_{pri}$ are consistent.

### 6.2.2 Association Cover, Privacy, and Utility

Consider a base table $T$, two attributes $A_i, A_j$ of $T$, and a candidate view scheme $V = \{v_1, ..., v_k\}$ for publishing the data in $T$. We use association cover to measure privacy. For an association between $(A_i = i, A_j = j)$, the association cover is a collection of values of $A_j$ that will be associated with $A_i = i$ by the join of view tables in $V$. We adapt the definition of association cover in [94] to our setting:

**Definition 6.2.1 [Association Cover]** For a given view $V$, let $J$ be the join result of view tables in $V$. Then: (1) for a private association $\alpha = (ID=i, P=p)$, the association cover $C^\alpha_P = \Pi_P(\sigma_{QID_P}(J))$, (2) for a public association $\alpha (P_1 = p_1, P_2 = p_2)$, the association cover $C^\alpha_P = \Pi_{P_1}(\sigma_{P_2}(J))$, where $\Pi$ and $\sigma$ are the projection and selection operators, respectively.

The size of $C^\alpha_P$, denoted as $|C^\alpha_P|$, is the number of distinct values in $C^\alpha_P$. For example, from the join result of the two view tables $v_1$ and $v_2$ in Figure 6.2, the cover of the association $\alpha (Name=Alice, Disease=AIDS)$ equals $\Pi_{Disease}(\sigma_{Age=30, Sex=F, City=Vancouver}(v_1 \bowtie v_2)) = \{AIDS, Diarrhea\}$. Therefore, from the join result of $v_1$ and $v_2$, both AIDS and Diarrhea are associated with Alice. We call those associations that are present in the cover but do not exist in the original base table noise. In the above example, $(Name=Alice, Disease=Diarrhea)$ is the noise. If the cover of the association is of size $k (k > 1)$, without any prior knowledge, the attacker cannot distinguish the true association from the noise, i.e., he/she cannot explicitly decide which association exists in the original base table. Thus, although we do not generalize the data, the association cover that is of size of $k$ has an effect similar to $l$-diversity.
For public associations, the association cover can also be used for utility purposes. It is straightforward that if the cover of public association from the views is the same as from the base table (i.e., free of noise), then the published view preserves 100% utility. Based on this, we define the safe and faithful view scheme as follows:

**Definition 6.2.2 [Safe and Faithful View Scheme]** Given a base table $T$, a set of private associations $\Sigma_{pri}$, a set of public associations $\Sigma_{pub}$, and a positive integer $k$, we say a view scheme $V$ is **safe** if for every private association $\alpha \in \Sigma_{pri}$, $|C_\alpha| \geq k$. We say a view scheme $V$ is **faithful** if for every utility association $\alpha \in \Sigma_{pub}$, $|C_\alpha| = |\Pi_{P_j}(\sigma_{P_i}(T))|$. 

For a specific private association $\alpha$, different view schemes may yield different degrees of privacy in terms of the size of association covers. For example, for the published view in Figure 6.2, the cover of association $\alpha(\text{Name}=\text{Alice}, \text{Disease}=\text{AIDS})$ is of size 2. However, for the published view in Figure 6.3, the cover of $\alpha$ is of size 1. In this chapter, we are interested in the safe and faithful view schemes that contain minimum noise. We allow users to specify a security threshold for private associations. If a view satisfies the threshold, we consider that as enough protection, which gives us more freedom in searching for a view with maximum utility. The formal definition is given below.

**Definition 6.2.3 [Optimal Safe and Faithful View Scheme]** Given a base table $T$ and a set of associations $S = \Sigma_{pri} \cup \Sigma_{pub}$, we say a safe and faithful view scheme $V$ is optimal if $\Sigma_{1 \leq i \leq |S|} |C_{\alpha_i}|$ is minimum, where $\alpha_i \in S$. 

In this chapter, we study the following problem:

**Problem statement** Given a base table $T$, a set of private associations $\Sigma_{pri}$, a set of public associations $\Sigma_{pub}$, and a positive integer $k$, find an optimal safe and faithful view scheme $V$. By seeking an optimal solution, we minimize the noise added to all associations beyond whatever is necessary as a result of guaranteeing the input threshold $k$. 

<table>
<thead>
<tr>
<th>Sex</th>
<th>City</th>
<th>Disease</th>
<th>History</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>Vancouver</td>
<td>Diarrhea</td>
<td>Cold</td>
</tr>
<tr>
<td>F</td>
<td>Toronto</td>
<td>Heart Attack</td>
<td>Stroke</td>
</tr>
<tr>
<td>M</td>
<td>Vancouver</td>
<td>AIDS</td>
<td>Cold</td>
</tr>
<tr>
<td>M</td>
<td>Alberta</td>
<td>Heart Attack</td>
<td>Stroke</td>
</tr>
<tr>
<td>30</td>
<td>Vancouver</td>
<td>Diarrhea</td>
<td>Cold</td>
</tr>
<tr>
<td>45</td>
<td>Vancouver</td>
<td>Heart Attack</td>
<td>Stroke</td>
</tr>
<tr>
<td>30</td>
<td>Toronto</td>
<td>AIDS</td>
<td>Cold</td>
</tr>
<tr>
<td>45</td>
<td>Vancouver</td>
<td>Heart Attack</td>
<td>Stroke</td>
</tr>
<tr>
<td>30</td>
<td>Alberta</td>
<td>Diarrhea</td>
<td>Cold</td>
</tr>
</tbody>
</table>

$V_1 = \Pi_{Sex,City}(T)$

$V_2 = \Pi_{Age,City,Disease,History}(T)$

Figure 6.3: Example of Unsafe Views
6.3 Faithful View Schemes

To find a safe and faithful view scheme, we start with faithful view schemes. It is straightforward that if we put two attributes in the same view table, the associations between the data values of these two attributes are always preserved. We make use of this property to design faithful view schemes. The approach is quite simple: for any public association on attributes $A_1$ and $A_2$, we put $A_1$ and $A_2$ in the same view table. Since every public association is preserved, the view that is constructed by this approach is faithful. However, it is possible that it is not safe. Before we explain the reason, we first define the critical attribute.

**Definition 6.3.1** [Critical Attribute] Given a base table $T$, let $\alpha = (ID=i, P=p)$ be any private association. Let $QID^\alpha$ be the QID values of $\alpha$. If there exists an attribute $A \in \text{schema}(T)$ such that there exists a value $v \in \Pi_A(QID^\alpha(T))$ that occurs exactly once in $T$, then $A$ is a critical attribute of $\alpha$, and $v$ is a critical value. We call the base table tuples that contain critical values critical tuples, and any attribute set that contains at least one critical attribute a critical attribute set.

Using the base table in Figure 6.1 as an example, for the association $\alpha = (Name=Alice, Disease=AIDS)$, $QID^\alpha = \{\text{tuple} \ 3\}$. Since the value Toronto of the attribute City has a unique occurrence in the base table, the value Toronto is a critical value and the attribute City is a critical attribute of $\alpha$. Any attribute set that contains City, e.g., $\{\text{City}, \text{Age}\}$, is a critical attribute set of $\alpha$. Note that whether an attribute is critical is determined by the private associations. If we change the association to be $(Name=John, Disease=Diarrhea)$, the City attribute is no longer a critical attribute.

Based on the definition of critical attribute, we have:

**Lemma 6.3.1 (Critical-attribute Property):** Given a base table $T$, let $\mathcal{V}$ be a view scheme, and $\alpha$ be a private association. If the join attributes in $\mathcal{V}$ that is used to reconstruct $\alpha$ only consist of critical attributes with regard to $\alpha$, then $|C_\mathcal{V}| = 1$.

The correctness of Lemma 6.3.1 is based on the property of critical attributes. Since the values that are used for join have unique occurrence in $T$, by joining on those values, no association noise will be constructed, which will result in the association cover of size 1. For example, for the view tables $v_1$ and $v_2$ in Figure 6.3, since they join on the critical attribute City, the size of the cover of association $(Name=Alice, Disease=AIDS)$ is 1.

The existence of critical attributes may result in that none of the faithful views are safe. The reason is shown by Lemma 6.3.2. In the lemma, for two associations $\alpha'$ and $\alpha''$, we use $\alpha' \cap \alpha''$ to specify the attribute that appears in both of them. For example, for two public associations $\alpha' = (\text{City}, \text{Disease})$ and $\alpha'' = (\text{City}, \text{Age})$, $\alpha' \cap \alpha'' = \text{City}$. We have:

**Lemma 6.3.2 (Faithful and Unsafe Views):** Given a base table $T$, a set of private associations $\Sigma_{pri}$ and a set of public associations $\Sigma_{pub}$, if: (1) the
union of attributes in the public associations contains all QID attributes and sensitive attributes, and (2) there exists a private association \( \alpha \in \Sigma_{\text{pri}} \) such that for every two public associations \( \alpha', \alpha'' \in \Sigma_{\text{pub}} \), \( \alpha' \cap \alpha'' \) is a critical attribute (set) of \( \alpha \), then there does not exist a faithful and safe view scheme.

**Proof:** We prove it by using contradiction. Assume the two conditions in the lemma are true, and there exists a faithful and safe scheme. Assume there are \( n \) view tables in the scheme, each of which preserves at least one public associations. Then following Lemma 6.3.1, by joining of these \( n \) view tables, the cover size of the private association is always 1, which contradicts the assumption that it is a safe scheme.

We assume security has higher priority to utility. Thus when we cannot have a safe and faithful view scheme from the base table, we have to sacrifice some of utility for security when we publish the view for data analysis. Thus we can do the following: the tuples that are critical to any private association will be moved from the base table for the design of the view scheme, so that the published views are always safe. Those removed tuples will be recorded by the data owner and be compensated for utility later after he/she has the data analysis result obtained from the published view. For example, assume Tim's record is removed from the base table in Figure 6.1 (since it contains a critical value Edmonton) and a view is constructed after the removal. Assume the public association is \( \text{City=?}, \text{Disease=Diarrhea} \). Then besides the analysis result from the published view, which will be \text{City=Vancouver}, the data owner will add \text{City=Edmonton}. By such compensation, the data analysis result is still the same as from the base table. In other words, by putting the attributes that are specified in a public association in the same view table, it can always achieve 100% utility.

### 6.4 Connectivity Graph

After we discussed the faithful views, we come to safe views. To decide whether a view is safe, size of cover of every private association has to be measured. The naive approach to measurement is to first materialize the view tables, then join those materialized view tables, and finally calculate the size of cover from the join result. It is straightforward that due to expensive view materialization and join operations, such approach is costly. Thus we use the **connectivity graph** to help us to efficiently measure the size of association covers without any view materialization. In this section, we first show our adapted definition of the connectivity graph (Section 6.4.1). Then we show how to efficiently measure the size of association covers by using the connectivity graph (Section 6.4.2).

#### 6.4.1 The Concept

Since the ID attribute is removed from the published data, in this chapter, we adapt the definition of the connectivity graph in Chapter 5. For a base table \( T \),
the connectivity graph $G(V, E)$ is constructed as follows: for every distinct data value in $T$, there is a corresponding node in $V$. For any two nodes $n_1$ and $n_2$ in $V$, there is an edge between them if there exists a tuple in $T$ that contains both values that $n_1$ and $n_2$ correspond to. Figure 6.4 shows part of the connectivity graph that is constructed based on the first and the third tuples in the base table (Figure 6.1). Compared with Chapter 5, besides the ID attribute and the sensitive attributes, the connectivity graph in this chapter consists of all QID attributes.

![Figure 6.4: The Example of Connectivity Graph](image)

**Definition 6.4.1 [Join Path]** Given a base table $T$, a view scheme $V$ and an association $\alpha(ID=i, P=p)$, let $J$ be the result of $v_1 \bowtie v_2 \bowtie \cdots \bowtie v_n$, where $\{v_1, \ldots, v_n\} \subseteq V$. We call $v_1 \rightarrow Jv_1\rightarrow v_2 \rightarrow Jv_2\rightarrow \cdots v_{n-1} \rightarrow Jv_{n-1}\rightarrow v_n$ a *join path* of $\alpha$ if $J$ only contains QID attributes and the attribute $P$ in $\alpha$. We call $Jv_i, v_{i+1}$ a *join label*.

For example, for the association $\alpha (Name=Alice, Disease=AIDS)$ and the view $V\{v_1 = \Pi_{Age,Sex}(T), v_2 = \Pi_{Sex,City}(T), v_3 = \Pi_{City,Disease}(T), v_4 = \Pi_{Disease,History}(T)\}$, the join path is $v_1 \rightarrow Sex \rightarrow City \rightarrow v_3$. Note that $v_4$ is not included since it is not needed for the reconstruction of $\alpha$.

Let $V = \{v_1, \ldots, v_k\}$ be a view scheme. It is *connected* provided the graph with $v_1, \ldots, v_k$ as nodes and containing an edge $(v_i, v_j)$ whenever $v_i, v_j$ share attributes, is connected. It is a *chain view scheme* provided this graph consists of a path or a set of disconnected paths. The challenge of non-chain view schemes is that branches and cycles may exist in the join graph of a view candidate. We must consider all possible join paths and find the minimum size of association covers over all such paths, to measure the privacy of an association. This is the added complexity. In this chapter, we confine attention to chain view schemes.

Given a view scheme, there may be multiple join paths. For example, for the view $V\{v_1 = \Pi_{Sex,City}(T), v_2 = \Pi_{Age,City}(T), v_3 = \Pi_{Age,Disease,History}(T)\}$, the join path for the association $(Sex=F, Age=30)$ can be either $v_1 \rightarrow City \rightarrow v_2$ or $v_1 \rightarrow City \rightarrow Age \rightarrow v_3$. Of these, the former is the shortest and has potentially the largest association cover for $(Sex=F, Age=30)$. However, since the second path includes an extra join on attribute $Age$, which is present in the association, the association cover for the longer path turns out to be the same as for the shorter path. In the following, by *the* join path for an association (with regard
Lemma 6.4.1 (One Join): Given a base table \( T \) and a view scheme \( \mathcal{V} \), let \( CG \) be the corresponding connectivity graph of \( T \). Then for the association \( (A_i = a_i, A_j = a_j) \) and view tables \( v_1, v_2 \in \mathcal{V} \), let \( n_i \) and \( n_j \) be the nodes in \( CG \) that correspond to \( A_i = a_i \) and \( A_j = a_j \). Then \( (A_i = a_i, A_j = a_j) \in v_1 \iff v_2 \) if and only if for every join attribute \( A \in J^{v_1,v_2} \), there exists a path \((n_i, m, n_j)\in CG\), where \( m \) corresponds to \( A = a \ (a \in \text{dom}(A)) \).

The correctness of Lemma 6.4.1 is based on the join operations and the properties of connectivity graph. Using the connectivity graph in Figure 6.4 as an example, node 5 corresponds to \( \text{City} = \text{Toronto} \), and node 8 corresponds to \( \text{Disease} = \text{Diarrhea} \). For the association \( (\text{City} = \text{Toronto}, \text{Disease} = \text{Diarrhea}) \), and the view \( \mathcal{V} = \{v_1 = \Pi_{\text{Age}, \text{Sex}, \text{City}}(T), v_2 = \Pi_{\text{Age}, \text{Sex}, \text{Disease}, \text{History}}(T)\} \), \( \alpha \in v_1 \iff v_2 \), since for join attributes \( \{\text{Sex}, \text{Age}\} \), there are paths \( \{5, 4, 8\} \) and \( \{5, 3, 8\} \) in the connectivity graph, where node 4 corresponds to \( \text{Sex} = F \) and node 3 corresponds to \( \text{Age} = 30 \).

Based on Lemma 6.4.1 we have:

Lemma 6.4.2 (Multiple Joins): Given a base table \( T \), let \( CG \) be the corresponding connectivity graph of \( T \). Let \( \mathcal{V} \) be any view scheme and \( \alpha \) be an association \( (ID = i, P = p) \). Let \( J = v_1 \rightarrow^{J^{v_1,v_2}} v_2 \ldots v_{n-1} \rightarrow^{J^{v_{n-1},v_n}} v_n \) be the join path of \( \alpha \). Let \( t \) be the node in \( CG \) that corresponds to \( P = p' \). Then \( (P = p') \in Cy \) if and only if for every \( (QID_i = qidi) \in QID^\alpha \), one of the following holds: (i) \( CG \) contains an edge \((s, t)\), where \( s \) corresponds to \( QID_i = qidi \); or (ii) \( CG \) contains a path \((s, q_1, q_{n-1}, t)\), where \( s \) corresponds to \( QID_i = qidi \), \( q_i \) corresponds to \( (A_i = a_i) \), and \( q_{i+1} \) corresponds to \( (A_{i+1} = a_{i+1}) \), such that \( (A_i = a_i, A_{i+1} = a_{i+1}) \in v_i \iff v_{i+1} \) by Lemma 6.4.1.

Intuitively \( P=p' \) is associated with \( ID=i \) if it is associated with every \( QID \) value that corresponds to \( ID=i \). Lemma 6.4.2 shows that such association can be constructed if \( P=p' \) is associated with every \( QID \) value by either the base table if they are in the same view table (the edge case), or a chain of joins if they are not (the path case). Using the connectivity graph in Figure 6.4 as an example, for the association \( \alpha (\text{Name} = \text{Alice}, \text{Disease} = \text{AIDS}) \) and the view \( \mathcal{V} = \{v_1 = \Pi_{\text{Name}, \text{Disease}}(T), v_2 = \Pi_{\text{City}, \text{Sex}}(T), v_3 = \Pi_{\text{Sex}, \text{Disease}, \text{History}}(T)\} \), the join path is \( v_1 \rightarrow^{\text{City}} v_2 \rightarrow^{\text{Sex}} v_3 \). Since through this join path, for every attribute in \( QID^\alpha = \{\text{Age} = 30, \text{City} = \text{Toronto}, \text{Sex} = F\} \), \( \text{Disease} = \text{diarrhea} \) is associated with \( \text{Age} = 30 \) (path \( \{3, 5, 4, 8\} \)), with \( \text{City} = \text{Toronto} \) (path \( \{5, 4, 8\} \)), and with \( \text{Sex} = F \) (edge \( \{4, 8\} \)). Then by Lemma 6.4.2, \( \text{Disease} = \text{diarrhea} \) is in the cover of association \( \alpha (\text{Name} = \text{Alice}, \text{Disease} = \text{AIDS}) \).

Based on Lemma 6.4.2 we have:

Theorem 6.4.1 (Size of Association Cover): Given a base table \( T \) and a view \( \mathcal{V} \), let \( CG \) be the corresponding connectivity graph of \( T \). Let \( \alpha (ID=i, P=p) \) be an association, then the size of the association cover \( |Cy| \) equals the number of \( P = p' \) that satisfies Lemma 6.4.2.
Function ACSize()

Input: (1) the connectivity graph G, (2) the association α(ID = i, P = p)
Output: the size of association cover of α

1. Let J be the join path of α.
2. size:=0;
3. ∀ node s ∈ G that corresponds to P = p
4. ∀(QIDi = qidi) ∈ QIDp
5. isIn:=true;
6. Let t ∈ G correspond to (QIDi = qidi).
7. if QIDi and P in same view table
8. {If(edge(s, t)) isIn:=false;
9. Else if !connected(QG, s, t, J) isIn:=false;
10. If(isIn) size++;
11. return size;

Procedure connected()

Input: graph G, source s, destination d, join path J
Output: return true if s is connected to d on path J, else false

12. i=0;
13. ∀ vertex u in G, set visited[u]:=false;
14. Push s to Stack S.
15. While CS is not empty
16. u:= pop S
17. If(u==t) {return true;
18. Else if 3d ∈ G such that not visited[d] and ∀ attribute A ∈ J[i],
19. 3 a corresponding node n ∈ G such that edge(u, n) ∧ edge(n, d)
20. Then {visited[d]:=true;
21. push d into S
22. i++;
23. Return false;

Figure 6.5: Pseudo Code: ACSize()

Theorem 6.4.1 shows that the problem of finding the size of association cover is equivalent to the problem of counting number of specific nodes that satisfy some properties in the connectivity graph. The correctness is straightforward from the definition of association cover and Lemma 6.4.2.

6.4.2 Traversing Connectivity Graph

For a specific association α(ID=i, P=p) and a given connectivity graph, we use the pseudo code in Figure 6.5 to determine the size of its association cover. The basic idea of the measurement is that for each value p' on attribute P, we check whether it is connected with every QID value by either existing in the same view table (Line 7-8) or join (Line 9). If it is true that P=p' is connected, the size of the association cover is increased by 1 (Line 10). The procedure connected() is used to check the connectivity by join. It is similar to depth-first search: we use the stack S to record the nodes that are traversed. Every node that is traversed as well as the nodes it is connected to are pushed into the stack (Line 18 - 21). The connectivity graph is traversed by popping the nodes in the stack (Line 16). Finally the size of the association cover is returned (Line 11).

We have:

Theorem 6.4.2 (Complexity of Measurement of Size of Association Cover) : Given a base table T, its corresponding connectivity graph G, and an association α, the complexity of Function ACSize() (Figure 6.5) that returns
the size of the cover of \( \alpha \) is \( O(|G|) \), where \( |G| \) is the size of the connectivity graph \( G \).

**Proof:** It is straightforward that the complexity of Function ACSSize() is decided by the procedure connected(). It is easy to show the complexity of the procedure connected() is the same as that of depth-first search, i.e., \( O(|G|) \).

It is important to note that for a given base table, the connectivity graph is unique, no matter what the private associations or the view schemes are. Our experimental results show that comparing with the joins-of-materialized-view approach, the connectivity graph approach affords a significant improvement in the performance of determination of association cover sizes. More details can be found in Section 6.6.5.

### 6.5 View Lattice

As we have seen, given a base table and a set of (public and private) association constraints, there are a large number of choices of views to publish. Different view schemes may yield different amounts of privacy and utility in terms of size of association covers. We first introduce view lattice as a convenient representation of all candidate view schemes (Section 6.5.1). Second, we study the properties of the view lattice (Section 6.5.2). We show how to use those properties to optimize the search procedure so that a safe and faithful view scheme can be efficiently found (Section 6.5.3).

#### 6.5.1 The Concept

The view lattice is a representation of the candidate view schemes as well as their relationships. For every view scheme candidate, there is a corresponding node in the view lattice \( \mathcal{L} \). We will describe the exact set of view schemes that form the nodes of \( \mathcal{L} \) operationally below. We first define the edges. For any two candidate view schemes \( \mathcal{V} = \{v_1, v_2, \ldots, v_m\} \) and \( \mathcal{V}' = \{v'_1, v'_2, \ldots, v'_n\} \), there is an edge from the corresponding node of \( \mathcal{V} \) in the lattice, say \( n_1 \), to that of \( \mathcal{V}' \), say \( n_2 \), if and only if: (1) there is only one view table \( v_s \in \mathcal{V} \) that has no matching view table in \( \mathcal{V}' \), (2) for all \( v'_i \in \mathcal{V}' \) that has no match in \( \mathcal{V} \), schema\( (v'_i) \subseteq \) schema\( (v_s) \), where \( v_s \) is the same as defined in condition (1). For two such view candidates \( \mathcal{V} \) and \( \mathcal{V}' \), we say that \( \mathcal{V} \) precedes \( \mathcal{V}' \), and \( n_1 \) is the parent of \( n_2 \) in the lattice. Using the view lattice in Figure 6.6 as an example, the view scheme that node 7 corresponds to precedes the view schema that node 5 corresponds to. This is because the view \( \{City, Sex\} \) in node 7 is the only one that has no matching schema in node 5. Furthermore, for the schemas in node 5 that have no matching in node 7 (in this example, \( \{Sex\} \)), is a subset of \( \{City, Sex\} \). Note that if there exists a view scheme \( \mathcal{V}'' \) such that \( \mathcal{V} \) precedes \( \mathcal{V}' \) and \( \mathcal{V}' \) precedes \( \mathcal{V}'' \), we will consider \( \mathcal{V} \) as the ancestor to \( \mathcal{V}'' \) and there is no direct edge between their corresponding nodes in the lattice. For example,
by expanding the second scheme in node 5, we also can get \( \{\text{Sex, Age, City, Disease, History}\} \) (corresponding to node 9 in Figure 6.6). Since node 9 is the ancestor to node 5, there is no direct edge between node 9 and node 5 in the lattice.

We now describe the nodes in \( \mathcal{L} \) operationally. First, there is only one node in the bottom of lattice. The schemes of view tables in this node are designed by the following rules: (1) The \( ID \) attribute is removed and will not be put into any view table. (2) For any two attributes that are involved in the same public association, they are put into the same view table in the bottom node. (3) For all remaining attributes that are not \( ID \) and not specified in any public association, they are put into different view tables, each containing only one attribute. It is straightforward that by following the three rules above, the bottom node represents a faithful view scheme. Using the associations in Example 6.2.1 as example, since the attributes \( \text{Age} \) and \( \text{Disease} \) are specified in the same public association, they are put together in the same view table in the bottom node of the lattice. Similarly for \( \text{Disease} \) and \( \text{History} \). The rest of non-\( ID \) attributes \( \text{City} \) and \( \text{Sex} \) are separately put into two view tables. The resulting bottom node is shown in Figure 6.6.

The construction of the lattice proceeds in a bottom-up fashion. Starting from the bottom node (whose level equals 1), we construct level \( k+1 \) from level \( k \) in the lattice. To be more specific, for every node \( p \) on level \( k \) of the view lattice, which consists a set of view tables \( \{v_1, \ldots, v_n\} \), we randomly pick a view table \( v_i \) and construct a new scheme \( v'_i \) from by expanding its schema with an attribute \( A \) that does not exist in \( v_i \). By replacing \( v_i \) with the new \( v'_i \) and deleting the view tables whose schemes are contained in \( v'_i \), a new view scheme is constructed which corresponds to a new node in the lattice. After all nodes \( Q \) on level \( k+1 \) are constructed by expansion, the nodes \( q \in Q \) that are the ancestors to any node on level \( k \) will be eliminated. Lastly, the edges between the node \( q \) on level \( k+1 \) and the node \( p \) on level \( k \) will be added, if \( q \) precedes \( p \). For example, if the view table \( \Pi_{\text{City, Sex}}(T) \) in the scheme that node 4 in Figure 6.6 corresponds to is expanded, we get node 6; otherwise, we get node 7 when the view table \( \Pi_{\text{Age, Disease, History}}(T) \) is expanded.

Note that the size of the lattice is exponential in the number of attributes in the scheme. For every node in the view lattice, except the top and the bottom, it may have multiple children and multiple parents (e.g., the node 4 in Figure 6.6). The top of the view lattice corresponds to the view scheme that contains only one table, which consists of all attributes except the \( ID \) attribute in the base table. It is straightforward that by such construction, every node in the lattice corresponds to a faithful view scheme. However, it may not be safe. Therefore, the size of the association covers has to be measured on each node in the lattice.

Note that the view lattice is simply a concept. We never materialize the whole view lattice when we search for the optimal safe and faithful view scheme. Instead, we only construct a small set of nodes in the lattice on the fly. More discussions of how we use the concept of view lattice to construct the final output view scheme can be found in Section 6.5.3.
Chapter 6. Privacy and Utility Preserving Data Publishing by Views

Figure 6.6: Part of View Lattice

To find the optimal safe and faithful view scheme, the naive approach is to measure the size of association cover for every private association on every node in the lattice. Since the number of nodes in the view lattice may be enormous, this approach is very costly. In fact, in the worst case, finding an optimal faithful and safe view scheme is NP-hard (shown in Theorem 6.5.1).

**Theorem 6.5.1 (NP-hard Complexity):** Given a base table with scheme $S$, a set of private associations $\Sigma_{pri}$, and a set of public associations $\Sigma_{pub}$, the complexity of finding an optimal safe and faithful view scheme $V$ is NP-hard.

**Proof:** We prove the NP-hard complexity result, the complexity result of which is in the number of attributes in the base table. The proof is by reduction from the knapsack problem. The knapsack problem is: given a set of items, each with a cost and a value, determine the items to include in a collection so that the total cost of these items is less than some given cost and the total value is as large as possible. Consider the problem of finding a binary decomposition of the base table that yields a safe and faithful view and is optimal with regard to the noise added. We can reduce the knapsack problem to this, showing this is NP-hard. The idea is that adding an item to the knapsack corresponds to dropping the corresponding attribute from the set of join attributes of the binary view scheme. Since this is a restriction to the original problem, the theorem follows.

Due to the NP-hard complexity, we propose finding a local optimum as a heuristic solution. A local optimum is defined as follows:

**Definition 6.5.1 [Locally Optimal Safe and Faithful View Scheme]:** Given a base table $T$ and a set of associations $\Sigma_{pri} \cup \Sigma_{pub}$, let $V$ be a view scheme, and its corresponding node in the view lattice be $p$. Then we say $V$ is *locally optimal* if for every node $p'$ on the same paths as $p$ in the lattice, $p'$ corresponds to either an unsafe view scheme or a safe scheme $V'$ whose total size of association covers is larger than that of $V$.

By definition, a locally optimal safe and faithful view scheme is optimal only out of the candidates that are on the same paths in the lattice. It may not be optimal out of all candidates. The naive method of finding a locally optimal safe and faithful view scheme is to first construct the view lattice, randomly pick a
path in the lattice, then find the locally optimal safe and faithful view scheme by reasoning on all view candidates on the path in the lattice. However, since the construction of the lattice is costly, the cost of such method is very expensive. In the following sections, we first discuss some important properties of the view lattice (Section 6.5.2). Then we show how to use those properties to optimize our search strategy on the lattice to return the locally optimal view scheme in polynomial time (Section 6.5.3). Our experiment results show that the locally optimal solution is reasonably good, while it has much better performance than the global solution. The details of the experiment result can be found in Section 6.6.3.

6.5.2 Properties of View Lattice

6.5.2.1 Monotonicity

The first important property of the view lattice is the monotone property of the size of association covers along the paths in the lattice. Before we discuss the details, we first have:

Lemma 6.5.1 (Containment of Association Cover) : Given a base table $T$, let $V$ and $V'$ be two view schemes constructed from $T$. Let $\alpha$ be a private association. Let $JP$ ($JP'$, respectively) be the join paths of $\alpha$ by $V$ ($V'$, respectively). Then if for every join label $I' \subseteq J$, there exists a join label $I \subseteq J$ such that $I' \subseteq I$, then $C_\alpha^V \subseteq C_\alpha^{V'}$.

The correctness of Lemma 6.5.1 is from the mechanism of join operations: since every distinct value $v$ on a join label $l$ (e.g., $\{City, Age\}$) must contain at least one distinct value $v'$ on a subset of $l$, say $l'$ (e.g. $\{Age\}$), every association $(A_i = a_i, A_j = a_j)$ that is obtained by join on value $v$ of $l$ must also be able to be obtainable by the join on value $v'$ of $l'$. Thus the association cover by join on $l$ must be contained in that by $l'$. Then we have:

Lemma 6.5.2 (Containment of Join Attributes in Lattice) : Given a view lattice $L$, let $p$ and $q$ be two nodes in $L$, where $p$ is $q$'s parent. Let $V$ and $V'$ be the view schemes that $p$ and $q$ correspond to. Let $\alpha$ be an association and $JP$ (respectively, $JP'$) be the join paths of $\alpha$ by $V$ (respectively, $V'$). Then for every join label $l' \subseteq J$, there exists a join label $l \subseteq J$ such that $l' \subseteq l$.

The correctness of Lemma 6.5.2 is based on the structural property of the view lattice. Since for every view table in the child node, its scheme is either equal to or is contained in at least one scheme table in the parent node, every join attribute of the child must equal to or be contained in that of the parent.

Based on Lemma 6.5.1 and Lemma 6.5.2, we have the monotone property of the view lattice:

Lemma 6.5.3 (Monotone Property) : Given a base table $T$, let $L$ be the corresponding view lattice. Let $p$ and $q$ be two nodes in $L$, where $p$ is $q$'s parent. Let $V$ and $V'$ be the view schemes that $p$ and $q$ correspond to. Then for any association $\alpha$, $C_\alpha^V \subseteq C_\alpha^{V'}$, i.e., $|C_\alpha^V| \leq |C_\alpha^{V'}|$.
Chapter 6. Privacy and Utility Preserving Data Publishing by Views

130

The correctness of Lemma 6.5.3 is straightforward from Lemma 6.5.1 and Lemma 6.5.2. Intuitively Lemma 6.5.3 shows that for any specific association, the size of its association cover is monotonically decreasing in the view lattice bottom-up. By using this property, when we search for a locally optimal safe and faithful scheme, which has the minimum sum of size of association covers out of all candidates on the same paths, we do not have to construct the whole view lattice. Instead, we start from the bottom of the view scheme and search bottom-up, until we reach a scheme that is not safe. Out of the children of the found unsafe scheme, we pick the one that is not only safe but also of minimal total size of association covers as the output, which is always a locally optimal safe and faithful scheme.

6.5.2.2 Equivalence

Lemma 6.5.3 shows that the size of the association cover does not increase when traversing the lattice bottom-up. In fact, there are some nodes in the view lattice that always have the same size for association covers. We first formally define such “equivalence”.

Definition 6.5.2 [Equivalent Schemes] For two view schemes V and V', we say they are equivalent, denoted as V ≡ V', if for every association α, $C_\alpha^V = C_\alpha^{V'}$.

The equivalent views can be classified into two cases: equivalent by join and equivalent by functional dependency. We discuss each of them individually in the following.

Equivalence by join: If we represent the join path as a set of join labels, for example, for the view $V = \{ v_1 = \Pi_{A,B}(T), v_2 = \Pi_{B,C}(T), v_3 = \Pi_{C,D,E}(T) \}$, the join path $v_1 \rightarrow^B v_2 \rightarrow^C v_3$ for the association $(A, E)$ is represented as $\{B\}$, $\{C\}$, then we have:

Lemma 6.5.4 (Equivalence by Join): Given two view schemes V and V', if for every association $\alpha$, V and V' have the same set of join labels in the same order, then $V \equiv V'$.

The correctness of Lemma 6.5.4 comes from the fact that the join attributes are the key factors that decide the association covers. Note that the order of the join attributes matters regarding the equivalence of associations. For example, for the base table T shown in Figure 6.7, we assume that we have two view schemes $V = \{ \Pi_{A,B}(T), \Pi_{B,C}(T), \Pi_{C,D}(T) \}$, and $V' = \{ \Pi_{A,C}(T), \Pi_{B,C}(T), \Pi_{B,D}(T) \}$. For the association $(A=a_1, D=d_1)$, the set of join labels of V is $\{\{B\}, \{C\}\}$, and the set of join labels of V' is $\{\{C\}, \{B\}\}$. They have different orders of the join attributes, which results in different association covers: $|C_\alpha^V| = 3$, and $|C_\alpha^{V'}| = 4$.

Equivalence by functional dependency: when some functional dependencies are available, there exist some view schemes in the lattice that are equivalent in the size of association covers due to functional dependencies. Before we explain the details of this kind of equivalence, we first introduce some notations.
Chapter 6. Privacy and Utility Preserving Data Publishing by Views

A B C D
1 a1 b1 c1 d1
2 a2 b1 c2 d2
2 a2 b2 c1 d3
2 a2 b2 c3 d4

Figure 6.7: Example: Order of Join Labels

<table>
<thead>
<tr>
<th>ID</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>a1</td>
<td>b1</td>
<td>c1</td>
<td>d1</td>
</tr>
<tr>
<td>2</td>
<td>a2</td>
<td>b1</td>
<td>c2</td>
<td>d2</td>
</tr>
<tr>
<td>3</td>
<td>a1</td>
<td>b2</td>
<td>c2</td>
<td>d3</td>
</tr>
<tr>
<td>4</td>
<td>a2</td>
<td>b2</td>
<td>c3</td>
<td>d4</td>
</tr>
</tbody>
</table>

Figure 6.8: Functional Dependency

For two sets of attributes $J_1$ and $J_2$ such that $J_1 \subseteq J_2$, we use $J_2 - J_1$ to denote the attributes that are in $J_2$ but not in $J_1$. For example, for $J_1 = \{A, B\}$ and $J_2 = \{A, B, C\}$, $J_2 - J_1 = \{C\}$. We use $p \rightarrow q$ to denote the functional dependency between attributes $p$ and $q$. We assume the functional dependencies are available to the data owner when designing the view schemes.

We have shown in Lemma 6.5.2 that there exists a containment relationship between the join labels of the parent and child nodes in the lattice. Then we have:

Lemma 6.5.5 (Equivalence by Functional Dependency): Given a base table $T$, let $L$ be the corresponding view lattice. Let $p$ and $q$ be two nodes in $L$, where $p$ is $q$'s parent. Let $V$ and $V'$ be the view schemes that $p$ and $q$ correspond to. Then for every association $\alpha$, let $JP (JP', \text{respectively})$ be the join path of $\alpha$ by $V (V', \text{respectively}).$ If for every join label $l'$ on $JP$, there exists a join label $l$ on $JP'$ such that $l \subseteq l'$ and $l' \rightarrow (l - l')$, then $V \equiv V'$.

The correctness of Lemma 6.5.5 is based on the fact that since for every join attribute $v' \in V'$, $v' \rightarrow (l - l')$, every distinct value on $l' \rightarrow (l - l')$ must correspond to a value on $l - l'$. Thus for every value that is associated with $ID = i$ by join on $l'$, it must also be associated with $ID = i$ by $l$. We use an example to illustrate the correctness of Lemma 6.5.5. For example, for the base table shown in Figure 6.8, we assume it has the functional dependency $B \rightarrow C$. We have $V = \{v_1 = \Pi_{A,B,C}(T), v_2 = \Pi_{B,C,D}(T)\}$, and $V' = \{v'_1 = \Pi_{A,B}(T), v'_2 = \Pi_{B,C,D}(T)\}$. The node in the view lattice that corresponds to $V$ is a parent of that of $V'$. For the private association $(ID = 1, D = d_1)$, the join labels of $V$ and $V'$ are $l = \{B, C\}$ and $l' = \{B\}$. Then $l - l' = \{C\}$. Since $B \rightarrow C$, every distinct value on $\{B\}$ must correspond to a distinct value on $\{B, C\}$, thus the association covers are of the same size for both $V$ and $V'$.
Chapter 6. Privacy and Utility Preserving Data Publishing by Views

132

Function FindOutputView()
Input: the scheme $S$ for base table $T$, threshold $t$,
private association set $\Sigma_{pri}$, public association set $\Sigma_{pub}$
Output: a locally optimal safe and faithful view scheme.
1. Initialize $L=\{\}//L$: base
2. $V_{pr}(A_i=a_j, A_j=a_k) \in \Sigma_{pub}$.
3. $L=L \cup \{(A_i, A_j)\}$
4. $\forall$ attribute $A$ not appearing in $L$, $L=L \cup \{(A)\}$
5. Initialize CheckedV=$L$
6. Repeat{
7. $V$=BottomUpLattice($L$, $\Sigma_{pri}$, CheckedV)
8. Add $V$ to CheckedV;
9. If $\exists i$ such that $V.Csize[i]<t$ 
   //V corresponds to an unsafe view.
10. find $V$'s children $C$ in CheckedV.
11. output scheme in $C$ of minimal total size of AC.
12. } Until $P$'s scheme = $S - \{ID attribute\}$.

Procedure BottomUpLattice()
Input: (1) view scheme $V_k$=$\{v_1, \ldots, v_n\}$ on level $k$.
(2) private associations set $\Sigma_{pri}$.
Output: view scheme $V_{k+1}$ on level $k+1$
13. $V_{vi} \in V_k$ //construct all possible parents of $V_k$
14. Initialize $PN=\{\}$. //PN stores all parent nodes
15. Repeat{ Initialize $V = V^N$.
16. Pick an attribute $A_i \in \text{schema}(v_i)$
17. $\text{schema}(V.v_i) = \text{schema}(v_i) \cup A_i$.
18. Remove $v_i$ from $V$ s.t. $\text{schema}(v_i) \subseteq \text{schema}(V.v_i)$.
19. Add $V$ to PN.} Until every $A_i$ is used
20. For every view scheme $V \in PN$
   //eliminates ancestor nodes from PN
21. If $\exists V' \in PN$ such that $V$ precedes $V'$ (Def. of precede is in Sec 6.5.1){
22. Remove $V$ from PN.}
23. Else //decide its association cover size
24. If $\exists V' \in \text{CheckedV}$ such that $V \equiv V'$
   //Equivalence property: Lemma 6.5.4 and 6.5.5 {
25. $V.Csize = V'.Csize$.
   //Csize: the size of association cover.}
26. Else $\forall$ $[i] \in \Sigma_{pri}$
27. If Lemma 6.3.1 holds
28. $V.Csize[i]=1$. //critical-attribute property
29. Else $V.Csize[i]=\text{ACSize}()$ (Figure 6.5).}
30. Add $V$ to CheckedV.)
31. Return the scheme of the minimal $\Sigma Csize[i]$ in PN as $V^{k+1}$.

Figure 6.9: Pseudo Code: FindOutputView()
6.5.3 Optimization of Search Strategy

Based on the critical-attribute property (in Section 6.3), the monotone property, and the equivalence property (in Section 6.5.2), we optimize the search procedure so that we can efficiently find the locally optimal safe and faithful view scheme in polynomial time. The basic idea is that starting from the bottom of the view lattice, the lattice is traversed bottom-up, until some node \( n \) whose corresponding view scheme is not safe is reached. Out of all children of the node \( n \), we output the view scheme that is not only safe but also of minimal total size of association covers. The pseudo code is shown in Figure 6.9. In the following we discuss more details of the optimized search strategy.

**Base of lattice:** Function \texttt{FindOutputView()} shows how we construct the base of the view lattice. The base of the view lattice contains one single node, which corresponds to the view scheme that contains a set of view tables. To be more specific, first, for every public association \( \alpha \), a corresponding view table that consists of the two attributes that are specified by \( \alpha \) is constructed (Line 2 - 3). Second, for every attribute \( A \) that is not specified in any public association, a view table that only consists of \( A \) is constructed (Line 4). Using the base table in Figure 6.1 and the association constraints in Example 6.2.1 as the example, the base of the lattice is a node that contains the view tables \( \{ \Pi_{\text{City}}(T), \Pi_{\text{Sex}}(T), \Pi_{\text{Age,Disease}}(T), \Pi_{\text{Disease,History}}(T) \} \).

**Bottom-up construction:** As mentioned in Section 6.4.1, we mainly consider the class of view schemes that correspond to a set of chains. After the base of the lattice is constructed, we repeatedly construct the other nodes in the lattice bottom-up (Line 7), until the node that corresponds to a scheme that is not safe is reached. We output the view scheme that is not only safe but also of minimal total size of association covers (Line 10 - 11) as the final result. Procedure \texttt{BottomUpLattice()} shows how the nodes in the lattice are constructed bottom-up. Specifically, for the node \( p \) on level \( k \) of view lattice, which consists a set of view tables \( V^k = \{ v_1, \ldots, v_n \} \), all of its possible parents are constructed in the following way: first, for every view table \( v_i \in V^k \), its schema is expanded with a new attribute \( A \) (Line 16 - 17). By removing the view tables whose schemes are contained in the new scheme of \( v_i \) (Line 18), a new view scheme \( V \) that corresponds to a new node on level \( k + 1 \) in the lattice is created. We use the set \( \mathcal{PN} \) to store all such candidate nodes on level \( k + 1 \) (Line 19). Second, the nodes from \( \mathcal{PN} \) that are ancestors to the node \( p \) on level \( k \) (Line 21 - 22) are eliminated. For every remaining candidate node \( p \) in \( \mathcal{PN} \), if its corresponding scheme is equivalent to any view \( V' \) in \( \mathcal{CheckedV} \), which stores all lattice nodes that have been traversed, its association cover size set \( Csize \) is the same as that of \( V' \) (Line 24 - 25). Otherwise, we calculate the size of the cover for each private association: if \( V \) satisfies Lemma 6.3.1, its corresponding cover size is labeled as 1 (Line 27 - 28); otherwise, Function \texttt{ACSize()} (shown in Figure 6.5) is applied to calculate the association cover (Line 29). After the size is calculated, the node \( p \) is added to \( \mathcal{CheckedV} \) (Line 30). Finally, the view scheme that is of the minimal total size of association covers on level \( k + 1 \) is picked as the seed for the expansion of the level \( k + 2 \) (Line 31).
Chapter 6. Privacy and Utility Preserving Data Publishing by Views

Figure 6.10: Search in Lattice

Figure 6.10 shows an example of searching through the view lattice for a locally optimal safe and faithful view scheme. We consider the private association (Name='Alice', Disease='AIDS') and the public associations as defined in Example 6.2.1. We assume the privacy threshold \( k \) equals 2. First we construct the bottom node (node 1 in Figure 6.10) and calculate its association cover. Since it is of size 3, which is larger than the threshold, we keep constructing the nodes bottom-up. Nodes 1 and 2 are equivalent by Lemma 6.5.4, thus they yield the same size of the association cover, which equals 3. We calculate the sizes of association covers of node 3 and node 4 by calling Function ACsize() (Figure 6.5). The sizes of association cover of node 3 and 4 are monotonely decreasing from 3 to 2. Then we reach node 5. Unfortunately, since the join attribute City is a critical attribute, node 5 yields 1 as the size of the association cover, which is less than the privacy threshold 2. The search procedure terminates and returns the scheme that the child of node 5, i.e., node 4, corresponds to. It is straightforward that the output scheme is a locally optimal safe and faithful scheme. We have:

**Theorem 6.5.2 (Complexity of Finding Locally Optimal Safe and Faithful Scheme):** Given a base table \( T \), a set of private associations \( \Sigma_{pri} \), and a set of public associations \( \Sigma_{pub} \), let \( G \) be the connectivity graph of \( T \). Function FindOutputView() (in Figure 6.9) correctly returns a locally optimal safe and faithful view in time \( O(n^2 \times |G|) \), where \( n \) is the number of attributes of \( T \), and \( |G| \) is the size of \( G \).

**Proof:** The local optimality of the returned view schemes can be proved using the definition of locally optimal view schemes, the properties of the view lattice, and the algorithm (in Figure 6.9). For the worst case that no two lattice nodes are equivalent and the privacy threshold is very small, we have to construct a node on every level in the lattice. For each constructed node, we need to calculate its size of association covers. First, to reason on the number of levels, for the worst case, there is no public association. In this case, every table scheme in the bottom node contains a single attribute. Assume there are \( n \) attributes in the base table. Then for the attribute on the \( i \)th level in the lattice, it has \( n - i \) steps of expansion to reach the root, which contains one
Chapter 6. Privacy and Utility Preserving Data Publishing by Views

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th># of values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Age</td>
<td>Numeric</td>
<td>73</td>
</tr>
<tr>
<td>Workclass</td>
<td>Categorical</td>
<td>9</td>
</tr>
<tr>
<td>Education</td>
<td>Categorical</td>
<td>16</td>
</tr>
<tr>
<td>Marital Status</td>
<td>Categorical</td>
<td>7</td>
</tr>
<tr>
<td>Occupation</td>
<td>Categorical</td>
<td>15</td>
</tr>
<tr>
<td>Race</td>
<td>Categorical</td>
<td>5</td>
</tr>
<tr>
<td>Gender</td>
<td>Categorical</td>
<td>2</td>
</tr>
<tr>
<td>Native Country</td>
<td>Categorical</td>
<td>42</td>
</tr>
<tr>
<td>Salary</td>
<td>Categorical</td>
<td>2</td>
</tr>
</tbody>
</table>

Figure 6.11: Adult Dataset

The number of levels of the lattice equals \( \sum_{i=1}^{n} (n-i) = n(n-1)/2 \). Thus the number of levels of the lattice is at worst \( O(n^2) \). Since we have shown in Theorem 6.4.2 that the complexity of calculation of association cover size is \( O(|G|) \), the theorem follows from this.

6.6 Experiments

We ran several experiments to measure the performance of our approach and explored different factors that impact the performance. Our experiments measured: (1) the performance of finding a locally optimal safe and faithful view versus the optimal safe and faithful view, (2) the performance of finding a locally optimal safe and faithful view scheme with different parameters: different private and public associations and different privacy thresholds \( k \), (3) the size of the connectivity graph, (4) the savings of measurement of size of association covers using our connectivity graph versus using a brute force project-join computation, and (5) the savings by equivalence property. In this section, we describe our main results and provide analysis to our observations. Unfortunately, all related work on view publishing (e.g., [94], [72]) has no experimental results, while the work on k-anonymity has different public constraints as ours, making an empirical comparison with these previous works is difficult.

6.6.1 Experimental Setup

**Dataset:** We used the Adults database from the UC Irvine Machine Learning Repository\(^1\), which was comprised of data from the US Census. We chose nine attributes (shown in Figure 6.11) out of fourteen attributes of the dataset. There are 32561 records in Adult dataset.

**Private and public association constraints:** We defined a set of private and public association constraints on Adult database. The set of private association constraints consist of associations between ID and three sensitive attributes:

\(^1\)ftp://ftp.ics.uci.edu/pub/machine-learning-databases/adult
Age, Occupation and Salary. The public associations are: (1) (WorkClass, Education), (2) (Martial Status, Occupation), (3) (Race, Education), (4) (Education, Occupation), and (5) (WorkClass, Gender).

We implemented the algorithm in C++. The experiments are run on a 3.4GHZ Pentium 4 machine with 2GB memory. All values reported are the average of five trials after dropping the maximum and minimum for different workloads.

6.6.2 Size of Connectivity Graph

To measure the size of connectivity graph for different base tables, we created four variations of the original Adult data by choosing different nine attributes out of the fourteen attributes of the Adult database. All those variations have the same 32561 records in total. For the instance whose total number of distinct values is substantially smaller than the total number of tuples, we call it a dense instance, since the distinct values are densely covered by different tuples. Otherwise, we call it a sparse instance. Out of the four testing instances, the first three are dense and the last one is sparse. For each instance, we constructed the corresponding connectivity graph and measured its size. The result is shown in Figure 6.12. We first observed that the size of the connectivity graph highly depends on the total number of distinct data values. The larger the total number of distinct data values in the original database, the larger the size of the connectivity graph is. Second, the size of connectivity graph of dense instances is always a small fraction of that of the original instance. For example, for the first three instances that are dense instances, the size of their connectivity graphs is only 0.06% to 0.5% of that of the base table. Third, the size of connectivity graph of sparse instances increases substantially compared with that of dense instances. However, it is still a small portion of that of the base table. For instance, for the fourth instance which is a sparse one, the size of its connectivity graph is only 15% of that of the base table. Thus, no matter whether the instance is sparse or dense, the connectivity graph is always space efficient.

<table>
<thead>
<tr>
<th>Instance</th>
<th>Total # of Distinct Values in Base Table</th>
<th>Size of Graph</th>
<th>Size of Base Table</th>
<th>% of Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 (dense)</td>
<td>79</td>
<td>1.8KB</td>
<td>2.7MB</td>
<td>0.06%</td>
</tr>
<tr>
<td>2 (dense)</td>
<td>171</td>
<td>4.0KB</td>
<td>3.0MB</td>
<td>0.1%</td>
</tr>
<tr>
<td>3 (dense)</td>
<td>470</td>
<td>11.2KB</td>
<td>2.0MB</td>
<td>0.5%</td>
</tr>
<tr>
<td>4 (sparse)</td>
<td>21778</td>
<td>396.2KB</td>
<td>2.6MB</td>
<td>15%</td>
</tr>
</tbody>
</table>

Figure 6.12: Size of Connectivity Graph
6.6.3 Locally Optimal Solution vs. Globally Optimal Solution

We compared our locally optimal safe and faithful view scheme with the optimal safe and faithful view scheme. Since the optimal approach is time costly, we used a small subset of Adult database which only contained 1000 records for this experiment. We measured the approximation ratio of the total size of association cover. The approximation ratio is defined as following: let $s_l$ be the total size of association cover of locally optimal solution and $s_o$ be that of optimal solution, the approximation ratio is $r = s_l/s_o$. The experiment results showed that the approximation ratio is always less than 2. We also measured the performance for both locally optimum and optimal approaches. The experiments showed that the locally optimal approach is around five times faster than the optimal approach. Thus our locally optimal solution is reasonably good, while it has much better performance than the globally optimal solution.

6.6.4 Performance

We measured the performance of finding the locally optimal safe and faithful schemes with different configuration parameters: (1) different private and public associations, and (2) different privacy thresholds $k$ for safe schemes. In the next, we explain the details of our observations.

Various association constraints

To measure the performance in terms of different association constraints, we chose different subsets of private and public associations and used them as various input. From the result shown in Figure 6.13 (a), we first observed that the performance improved when the number of public associations increased. This is because when more public associations are available, more attributes are put together into the same table and fewer attributes are put alone in the bottom node in the lattice, which consequently results in fewer levels in the lattice that have to be traversed and fewer nodes that have to be constructed. Second, we observed that the more private associations, the worse the performance. This is because during the search of view lattice, on every node that is traversed, the size of the cover of every private association must be measured. The more private associations that are available, the more cost of the measurement is needed.

Privacy threshold $k$

We set different thresholds $k$, which is a positive integer, for requirement of safe views. Note that the number of distinct values of the sensitive attribute decides the upper bound of the threshold. Since the sensitive attribute Salary has only two distinct values, for this part of experiment, we mainly measured the performance on sensitive attributes Age and Occupation. We ran the experiments with various number of public associations. Figure 6.13 (b) shows the results when both attributes are considered as sensitive. The result shows that the larger the privacy threshold $k$ is, the better performance we have. This is because due to the monotone property of the lattice, the views that satisfy the
larger $k$ are always located at the lower level of the lattice. Thus by traversing the lattice in bottom-up fashion, the search always terminates earlier for larger threshold $k$. Similar results are observed for one-sensitive-attribute case. The results of the one-sensitive-attribute case are omitted for brevity.

### 6.6.5 Optimization

**Measurement of size of association cover: by connectivity graph vs. by joins**

The size of the association cover can be obtained from either our connectivity graph or the join result of the materialized view tables. We compared the performance of both approaches for different number of joins, and measured the saving ratio by our connectivity graph approach. Specifically, let $t_1$ be the cost of join approach, and $t_2$ be the cost of connectivity graph approach. We define the saving ratio as $R = (t_1 - t_2)/t_1$. Intuitively, the closer to 1 the $R$ is, the more savings we have. The joins were manipulated by running the SQL queries in MySQL. Figure 6.14 (a) showed that our connectivity graph approach is much faster than the join approach: when there is only one join, our approach saved around 97%. The saving ratio substantially increases when the number of joins increases. Starting from two joins, the saving ratio is very close to 1, which means the cost of the connectivity graph approach is a negligible fraction of that of the join approach. Therefore, our connectivity graph approach gets much better performance than the join approach.

**Effect of Equivalence Property**

We measured how much cost would be saved by respecting the equivalent property by join and by functional dependency. We specified three functional dependencies: Education $\rightarrow$ Occupation, Occupation $\rightarrow$ WorkClass, and Native-Country $\rightarrow$ Race. We adjusted the Adult data so that it satisfied these three functional dependencies. We measured the saving ratios when one, two, and three functional dependencies were available. The saving ratio is defined as: let $t_1$ be the time cost for the case that the lattice is traversed without any equivalence that is considered. Let $t_2$ be the time cost when the search strategy is optimized by taking equivalence property into consideration. We define the saving ratio as $R = (t_1 - t_2)/t_1$. Intuitively, the closer to 1 the $R$ is, the better savings we have. First, we observed that if we only consider equivalence by join, we always can reach no less than 40% of savings. Second, we observed that the more functional dependencies that were available, the better the savings we had. Figure 6.14 (b) showed the result of saving ratios by equivalence by functional dependency when there are two private associations available. For three functional dependencies, we saved more than 50% of time. This is because the more functional dependencies that are available, the more nodes in the lattice will be considered as equivalent. Thus the measurement of association cover size is not needed for those equivalent nodes. Similar experimental results are observed for one-private-association and three-private-association case. The results are omitted for brevity.
Chapter 6. Privacy and Utility Preserving Data Publishing by Views

6.7 Summary

Privacy and utility are two equally important concerns for publishing private data. In this chapter, we considered publishing the relational database by using views with protected privacy and preserved utility. We formally defined the measurement of both privacy and utility of the published views and proposed the connectivity graph as a space and time efficient means of measuring the size of association covers, which is used for gauging privacy and utility. For a given base table and a set of private and public associations, there may be a huge set of candidates for published view schemes. We proposed the view lattice to represent all candidates. We motivated the problem of finding an optimal view scheme that satisfies both privacy and public constraints and has minimum noise, and showed that finding the optimal solution is NP-hard. We proposed a heuristic as locally optimal safe and faithful view. Based on the properties of the view lattice and relationships between the view candidates, we optimized the search strategy and showed that we can efficiently find a locally optimal safe and faithful view scheme.
Chapter 7

Conclusions and Future Work

We have presented solutions for several security-related problems that have arisen due to data publishing in general. The set of problems considered fall into two areas: (1) efficient and secure query evaluation over published XML database, and (2) publishing relational databases for both privacy and utility.

We have made the following contributions with respect to efficient and secure query evaluation:

1. We first considered the problem of efficient and secure query evaluation over an access-controlled XML databases. We focused on an efficient algorithm that determines an optimal set of security check annotations with tree-pattern query nodes, by analyzing the subtle interactions between inheritance of security levels and the paths in the DTD graph. We experimentally validated our algorithm, and demonstrated the performance benefits to optimizing the secure evaluation of tree-pattern queries.

2. We have studied the problem of efficient and secure evaluation of XPath queries on (partially) encrypted XML databases hosted by an untrusted server. We formally defined the attack model and semantics of the security of our encryption scheme, metadata and query processing. We showed our approach of constructing a secure encryption scheme and showed that establishing an optimal secure encryption scheme is NP-hard. We proposed a secure metadata mechanism (including structural and value indices) at the server side that enables efficient query evaluation. We also proved that an attacker's belief probabilities about sensitive structural or value association information about the database is always very small, even after observing a series of queries and responses. Our experiments show that our techniques can deliver efficient query evaluation while ensure the enforcement of the security constraints.

Next, we consider the problem of publishing relational databases for both privacy and public constraints. We have:

- To answer the question of how to formally measure the privacy breach of the published views, we defined a general framework to measure the likelihood of a privacy breach of the published views by using probability. We proposed two attack models and for each attack model, we derived the formulas to calculate the probability of privacy breach.
• We extended the model to include utility. For a given relational base table and a set of private and public associations, there may be a huge set of candidates for published view schemes. We proposed the view lattice to represent all candidates. We motivated the problem of finding an optimal view scheme that satisfies both privacy and public constraints and has minimum noise, and showed that finding it is NP-hard. We proposed a heuristic as locally optimal safe and faithful view. Based on the properties of the view lattice and relationships between the view candidates, we optimized the search strategy and showed that we can efficiently find a locally optimal safe and faithful view scheme.

7.1 Future Work

Several problems still remain open. We mention a few of them in this section:

7.1.1 Efficient and Secure Query Evaluation

Access-controlled XML Database

Two of the many interesting questions spawned by this work are:

1. How does one deal with cyclic DTDs? DTD paths corresponding to a pair of query nodes may need to traverse DTD nodes multiple times to take care of various possibilities for overriding nodes, and the algorithms presented in this thesis are not directly applicable.

2. Can one design efficient security check optimization algorithms that utilize knowledge about database statistics to choose among multiple minimal SC annotations?

Encrypted XML Database

The following problems still need to be solved in this area:

1. The security achieved comes at the price of increase in key size and the need to decrypt data to answer some queries. Can we characterize the trade-off between security and efficiency?

2. Our current scheme cannot provide security against an attacker who has prior knowledge of tag distribution, query workload distribution, or correlations between data values. Extensions for handling these attack models are important.

3. Developing a secure encryption scheme for efficiently supporting updates is another important problem.

7.1.2 Privacy-preserving Data Publishing

1. We will continue to work on the k-view-table ($k > 2$) case and see whether instead of enumerating all possible worlds and interesting worlds, there exist more efficient approaches to computing the probability.
2. Our work of privacy view design problem mainly focused on restricted model. The question that is quite natural to ask is, can we extend our work to the *unrestricted model*?

3. It is an interesting extension to compare different utility models from the viewpoint of accuracy of data mining results obtained and the amount of noise introduced, while achieving a given level of privacy.
Bibliography


Bibliography


[87] Xiaokui Xiao, Yufei Tao, Anatomy: simple and effective privacy preservation. In International Conference on Very Large Databases (VLDB), pages 139 - 150, 2006.


Appendix A

Proofs: Secure and Efficient Query Evaluation over Access-controlled XML Databases

The proofs of the main theoretical results in Chapter 3 are given here. The DTD used in the examples is shown in Figure A.1.

Lemma 3.4.1: Completeness of PC Rule
Let $Q$ be a tree pattern query, $\Delta$ be any disjunctionless DAG DTD, and $(y,x)$ be an (ad- or pc-) edge in $Q$. Suppose $D$ is a valid database instance of $\Delta$. Then in every matching $\mu : Q \rightarrow D$, $\mu(x)$ and $\mu(y)$ have almost the same security levels if and only if one of the three cases (details of three cases are shown in Chapter 3) holds. □

Proof:
The "if" direction is straightforward: for any two query nodes, $x$ and $y$, that satisfy one of the three cases, by the analysis of each case (case I, case II, and case III), their corresponding nodes must have almost the same security level .

The "only if" direction is proved by contradiction: assume there exist two query nodes, $x$ and $y$, satisfy none of the three cases, but their matchings always have almost the same security levels. Then we show the assumption will result in contradiction.

We assume node $y$ is the query parent to node $x$. If none of the three cases stands, one of the following conditions must be true:

- Condition (1): there exists an overriding path from $y'$ to $x'$.
- Condition (2): every path from $y'$ to $x'$ is optional.

Then for each condition, we can create a corresponding instance as a counter example, which shows the violation of the condition will result in the failure of optimization. Figure A.2 shows one example for condition (1). In this example, since the path from "Accounts" to "Customer" is overriding (node "Account" is an overriding node), none of the nodes in the query Accounts//Customer whose SC annotation can be optimized. In fact, there exists an instance (partly shown in Figure A.2) such that each matching of the node Accounts is accessible, while each matching of the node Customer is not. Similarly the same result holds for
Condition (2). The details are omitted. Thus if there exist two query nodes, \( x \) and \( y \), that satisfy none of the three cases, their matchings cannot always have the same security levels, which contradicts the assumption. Therefore, the three cases are necessary for optimization of PC relation nodes.

**Lemma 3.4.2: Completeness of CC – Unconstrained**

Let \( Q \) be a tree pattern query, \( \Delta \) be a DTD, and \( x \) be an unconstrained node in \( Q \). Suppose \( x \) cannot be optimized using PC and CP rules. Then \( x \) can be optimized if and only if one of the conditions Case IV or Case V holds.

**Lemma 3.4.3: Completeness of CC – Constrained**

Let \( Q \) be a tree pattern query, \( \Delta \) a DTD, and \( x \) a constrained node in \( Q \). Suppose \( x \) cannot be optimized using PC and CP rules. Then \( x \) can be optimized if and only if Case V holds.

We prove the correctness of both Lemma 3.4.2 and 3.4.3 together.

**Proof:**

The “if” direction is straightforward: for any two query nodes \( x \) and \( y \) that satisfy case IV or case V, by the analysis of case IV and case V, their corresponding nodes must have almost the same security level.

The “only if” direction is proved by contradiction: assume there exist two query nodes, \( x \) and \( y \), that satisfy neither case IV nor case V, but their matchings always have almost the same security levels. We then show the assumption will result in contradiction.
Figure A.3: CC Rule for unconstrained nodes: (a) case IV.1; (b) case IV.2; (c) case IV.3.

Figure A.4: CC Rule for constrained nodes: (a) case V.1; (b) case V.2; (c) case V.3.

Suppose $x$ is unconstrained and $y$ is annotated. Let $w$ be the least common ancestor of $x$ and $y$ in the query tree and $S_x$ ($S_y$) be the set of ancestors of $x$ ($y$) occurring on the path from $w$ to $x$ ($y$) in the query tree, including $w$. If none of the two cases stands, one of the following conditions must be true:

- **Condition (1):** there does not exist a DTD path $P$ from $w'$ to $x'$ that passes through this overriding node $z'$ as well as through all nodes in $S_x'$, such that the segment of $P$ from $z'$ to $x'$ is non-optional and non-overriding (illustrated in Figure A.3(a)) (violates case IV.1).

- **Condition (2):** there does not exist a DTD $P$ from $w'$ to $x'$ that passes through all nodes in $S_x'$ such that the segment of $P$ from $y'$ to $x'$ is non-overriding and non-optional within the query context of $S_y'$ (illustrated in Figure A.3(b)) (violates case IV.2).

- **Condition (3):** $x'$ does not appear on every path $P$ in the DTD from $w'$ to $y'$, or the segment $P_1$ of $P$ from $x'$ to $y'$ is non-overriding (illustrated in Figure A.3(c)) (violates case IV.3).

- **Condition (4):** suppose $z'$ is a common ancestor of $x'$ and $y'$ in the DTD and furthermore if there are multiple paths $P$ from $w'$ to $y'$, or the segment $P_1$ of $P$ from $w'$ to $z'$ is not unique, or there exists some overriding nodes
between $z'$ and $y'$ or between $z'$ and $x'$ (illustrated in Figure A.4(a)) (violates case V.1).

- Condition (5): suppose $y'$ is an ancestor of $x'$ in the DTD and $z'$ is an ancestor of $y'$. If there are multiple DTD paths $P$ from $w'$ to $y'$ which pass through the query context nodes $T'$, or the segment $P_1$ of $P$ from $w'$ to $z'$ is not unique, or there exists overriding nodes between $z'$ and $x'$ (illustrated in Figure A.4(b)) (violates case V.2).

- Condition (6): suppose $x'$ is an ancestor of $y'$ in the DTD and $z'$ is an ancestor of $x'$. If there are multiple paths $P$ from $w'$ to $y'$ which passes through the context nodes $T'$, or the segment $P_1$ of $P$ from $w'$ to $z'$ is not unique (illustrated in Figure A.4(c))(violates case V.3).

Then for each condition we can create a corresponding instance as a counter example, which shows the violation of the condition will result in the failure of optimization. Figure A.5 shows an example for Condition (1). In this example, since the path from Accounts to Customer is overriding (node Account is overriding), and the path from node Accounts to PaymentInfo is optional (the edge between Account and PaymentInfo is labeled with "?"), node PaymentInfo in the query cannot have its SC annotation be optimized, since there exists an instance (partly shown in Figure A.2) such that each matching of node PaymentInfo is not accessible. Similarly we can create counter examples for the conditions (2) - (6). The details are omitted. Thus if there exist two query nodes $x$ and $y$ that satisfy none of the conditions Case IV or Case V holds, their matchings cannot always have the same security levels, which contradicts the assumption. Therefore, the condition that either Case IV or Case V holds is necessary for optimization of CC-relation nodes.

**Lemma 3.4.5 Remove CP(s)**

Let $Q$ be any query, $\Delta$ any disjunctionless DAG DTD, and $\Lambda$ the annotation assigning $(\text{RC})$ to all nodes of $Q$. For any (correct) annotation $\Lambda'$ obtained from $\Lambda$ by applying $\{\text{Root, PC, CP, CC}\}$, we can always reach a correct annotation $B$ by applying $\{\text{Root, PC, CC}\}$. We have either (i) $B <_{\text{RC}} \Lambda'$ or (ii) $B =_{\text{RC}} \Lambda'$. We have either (i) $B <_{\text{RC}} \Lambda'$ or (ii) $B =_{\text{RC}} \Lambda'$.

**Proof:**

We assume $y$ is the query parent of $x$. If $y$ can be optimized by CP rules, the matching of $y$ has almost the same security level of $x$, i.e., the matching of $x$
Lemma 3.4.6 Move PC(s) Ahead
Let $Q$ be any query, $\Delta$ be any disjunctionless DAG DTD, and $A$ be the annotation assigning (RC) to all nodes of $Q$. For any (correct) annotation $A'$ obtained from $A$ by applying \{(Root, PC, CC)\}*, we can always reach a correct annotation $B$ by firstly repeatedly applying the PC rule until there are no more nodes whose SC annotation can be optimized, then applying CC rule in the same fashion. We have either (i) $B \prec_{RC} A'$ or (ii) $B =_{RC} A'$ and $B \preceq_{LC} A'$.

Proof:
Before we give the details of the proof, we first define the equivalence of optimization operations.

Definition A.0.1 [Equivalence of Optimization Operations] We use $0(t)$ to represent the optimization on a query node $t$ by using the rule $O$, where $P \in \{PC, CC, CP\}$. We use $(\ldots, O_1(t_1), O_2(t_2), \ldots)$ to represent a sequence of the operations in which the operation $O_2$ that optimizes node $t_2$ follows the operation $O_1$ that optimizes node $t_1$. We say two sequences of operations $S$ and $S'$ are equivalent, annotated as $S \equiv S'$, if $S$ and $S'$ optimize the same set of nodes.

Then we prove the correctness of Lemma 3.4.6. Let $\sum = \{PC, CC, CP\}$. Let $A (A')$ be a sequence of operations that can be expressed by $\sum^* (PC^*CC^*)$. For two given query nodes $p$ and $q$, we want to show that

\begin{itemize}
  \item $(A, CC(q), PC(p), A') \equiv (A, PC(p), CC(q), A')$, i.e., the PC rule can always be applied before the CC rule.
  \item $(A, CP(q), PC(p), A') \equiv (A, PC(p), CP(q), A')$, i.e., the PC rule can always be applied before the CP rule.
\end{itemize}

The above can be proven by study of the details of PC, CC, and CP rules. Specifically, if node $p$ is an unconstrained node, if node $q$ that is optimized by $(A, CC(q), PC(p), A')$ is optimized by case IV or case V (the two cases for the CC rule), and the node $p$ is optimized by case II or case III (the two cases for the PC rule), by reasoning on each subcase we can see that the optimization on node $q$ can be moved after the optimization on node $p$.

If node $p$ is a constrained node, similarly, by reasoning on case II, case IV and case V, the node $q$ optimized by $(A, CC(q), PC(p), A')$ or by $(A, CP(q), PC(p), A')$ will not influence $p$'s optimization if we move $PC(p)$ ahead.

Theorem 3.4.2: Order of Optimization Rules
Let $Q$ be any query, $\Delta$ any disjunctionless DAG DTD, and $A$ the annotation assigning (RC) to all nodes of $Q$. Then the annotation $B$ obtained by applying $Root(\text{PC}^*)(\text{CC}^*)$ to query nodes, with the $\text{PC}^*$ and $\text{CC}^*$ closures applied top-down, is correct and minimal. Furthermore, for any (correct) annotation $A'$
obtained from \( A \) by applying \{Root, PC, CP, CC\}*, we have either (i) \( B \leq_{RC} A' \) or (ii) \( B =_{RC} A' \) and \( B \leq_{LC} A' \).

**Proof:**

Assume we have \( n \) PC(s) operations applied, \( n \) CC(s) operations applied, and \( t \) CP(s) operations applied in \{Root, PC, CP, CC\}*. We use \{\( P \)\}_x to annotate that the operations \( P \) are applied \( x \) times in sequence. Following Lemma 3.4.5 and Lemma 3.4.6 we have

\[
\{\text{Root,PC,CC,CP}\}_{m+n+t} = \text{RootPC}_m \{\text{CC,CP}\}_{n+t}; \quad \text{(Lemma 3.4.6)}
\]

\[
= \text{RootPC}_m \{\text{CC,PC}\}_{n+t}; \quad \text{(Lemma 3.4.5)}
\]

\[
= \text{RootPC}_{m+t} \text{CC}_n; \quad \text{(Lemma 3.4.6)}
\]

Note that first applying the PC rule makes the non-optimized node closer to the query root, while first applying the CP rule makes the optimized node farther away from the query root. Since the closer \( RC \) is to the query root, the less cost it has, for any (correct) annotation \( A' \) obtained from \( A \) by applying \{Root, PC, CP, CC\}*, we can obtain the annotation \( B \) by applying Root(PC*)(CC*) such that \( B \) cannot be more expensive than \( A \).

**Theorem 3.5.1: Completeness**

Let \( \Delta \) be a disjunction DTD and \( Q \) be a query. A node \( x \) in \( Q \) is optimizable if and only if it can be optimized using one of the modified rules Root, PC, CP, CC.

**Proof:**

The proof of the theorem is similar to the Theorem 3.4.1 for the disjunctionless DTD case. The only difference is the subtraction of the labeling. By studying the properties of labeling and each optimization rule, it is straightforward to show the correctness.

**Theorem 3.5.2: Order of Optimization Rules**

Let \( \Delta \) be a disjunction DTD and \( Q \) be a query. Let \( A \) be the annotation assigning \( (RC) \) to all nodes of \( Q \), and \( A' \) be any annotation obtained from \( A \) by applying the rules \{Root, PC, CP, CC\} in any order. Then there is an annotation \( B \) such that \( B \) is correct, \( B \leq A' \), and \( B \) can be obtained from \( A \) by applying the Root rule, then the PC rule repeatedly to saturation to nodes of \( Q \) top-down, followed by the CC rule repeatedly to saturation to nodes of \( Q \) in top-down fashion.

**Proof:**

The proof of the theorem is based on studying the details of the optimization rules for disjunction DTD case, which is similar to the Theorem 3.4.2 for the disjunctionless DTD case. The details are omitted.

Finally, we have:

**Theorem 3.5.3: Optimality**

Let \( \Delta \) be a disjunction DTD and \( Q \) be a query. Then a minimal correct annotation of \( Q \) can be obtained in time polynomial in the size of \( Q \) and \( \Delta \).

**Proof:**

The proof of the theorem is similar to the Theorem 3.4.3 for the disjunctionless
DTD case. The only difference is the definition and implementation of the "non-overriding non-optional path". The check NONOR($x', y'$) can be done in polynomial time by the following procedure: there exists a "non-overriding non-optional path" from node $x'$ to $y'$ if and only if: (i) $x'$ is an element node, its disjunction child is $c'$ and NONOR($c', y'$) holds; or (ii) $x'$ is a disjunction node and for each sequence child $s'$ of $x'$, NONOR($s', y'$) holds; or (iii) $x'$ is a sequence node and $\exists$ an element child $e'$ of $x'$ such that the DTD label on the edge ($x', e'$) is '1' or '+' and NONOR($e', y'$); or (iv) nodes $x'$ and $y'$ are identical. |