Brigham Young University
BYU ScholarsArchive

Theses and Dissertations

2006-05-09

Trust Negotiation for Open Database Access Control

Paul A. Porter
Brigham Young University - Provo

Follow this and additional works at: https://scholarsarchive.byu.edu/etd

Part of the Computer Sciences Commons

BYU ScholarsArchive Citation

Porter, Paul A., "Trust Negotiation for Open Database Access Control" (2006). *Theses and Dissertations*. 422.
https://scholarsarchive.byu.edu/etd/422

This Thesis is brought to you for free and open access by BYU ScholarsArchive. It has been accepted for inclusion in Theses and Dissertations by an authorized administrator of BYU ScholarsArchive. For more information, please contact scholarsarchive@byu.edu, ellen_amatangelo@byu.edu.

TRUST NEGOTIATION FOR OPEN DATABASE ACCESS CONTROL

by

Paul Alan Porter

A thesis submitted to the faculty of
Brigham Young University
in partial fulfillment of the requirements for the degree of

Master of Science

Department of Computer Science
Brigham Young University
August 2006

Copyright © 2006 Paul Alan Porter

All Rights Reserved

BRIGHAM YOUNG UNIVERSITY

GRADUATE COMMITTEE APPROVAL

of a thesis submitted by

Paul Alan Porter

This thesis has been read by each member of the following graduate committee and by majority vote has been found to be satisfactory.

Date	Kent E. Seamons, Chair
Date	David W. Embley
Date	Quinn O. Snell

BRIGHAM YOUNG UNIVERSITY

As chair of the candidate's graduate committee, I have read the thesis of Paul Alan Porter in its final form and have found that (1) its format, citations, and bibliographical style are consistent and acceptable and fulfill university and department style requirements; (2) its illustrative materials including figures, tables, and charts are in place; and (3) the final manuscript is satisfactory to the graduate committee and is ready for submission to the university library.

Date

Kent E. Seamons
Chair, Graduate Committee

Accepted for the Department

Parris K. Egbert
Graduate Coordinator

Accepted for the College

Thomas W. Sederberg
Associate Dean, College of Physical and Mathematical Sciences

ABSTRACT

TRUST NEGOTIATION FOR OPEN DATABASE ACCESS CONTROL

Paul Alan Porter

Department of Computer Science

Master of Science

Hippocratic databases are designed to protect the privacy of the individuals whose personal information they contain. This thesis presents a model for providing and enforcing access control in an open Hippocratic database system. Previously unknown individuals can gain access to information in the database by authenticating to roles through trust negotiation. Allowing qualified strangers to access the database increases the usefulness of the system without compromising privacy.

This thesis presents the design and implementation of two methods for filtering information from database queries. First, we extend a query modification method for use in an open database system. Second, we introduce a novel filtering method that overcomes some limitations of the query modification method. We also provide results showing that the two methods have comparable performance that is suitable for interactive response time with our sample data set.

ACKNOWLEDGMENTS

I would like to thank my advisor, Kent Seamons, for his help in writing and editing this thesis. Reed Abbott, Travis Leithead, Dan Walker, Tim van der Horst, and Nate Seeley have also been a great help. I would also like to thank my family for their support.

Table of Contents

1 Introduction 1

1.1 HIPAA and Privacy 2
1.2 Database Access Control 3
1.3 Trust Negotiation 3
1.4 Differences from Traditional Trust Negotiation 4
1.5 Motivating Scenario 5
1.6 Goals 5
1.7 Thesis Outline 6

2 Related Work 7

3 System Design 11

3.1 Policies 11
  3.1.1 Column Policies 11
  3.1.2 Row and Cell Policies 11
  3.1.3 Content-Based Policies 12
3.2 Trust Proxy 13
  3.2.1 Results Filtering 13
  3.2.2 Query Modification 15

4 Implementation 17

4.1 Policy Storage 17
4.2 Results Filtering Implementation 18
4.3 Query Modification Implementation 20

4.4 Example Scenario 22
4.4.1 Results Filtering 23
4.4.2 Query Modification 25
4.5 Comparison of Filtering Methods 26
4.6 Software 27

5 Performance Analysis **31**

6 Threat Model **35**

7 Conclusions and Future Work **39**

References **44**

List of Tables

3.1 Query results for the query `SELECT name, diagnosis, phone FROM patients WHERE diagnosis = 'cancer.'` Note the leaked diagnoses of Sally and Reed. 15

3.2 The modified query results 15

4.1 Cell-level diagnosis policies stored as role expressions 18

4.2 Cell-level telephone policies stored as separate permissions 18

4.3 Cell-level policies for diagnoses 21

4.4 Column Policies 23

4.5 Unfiltered query results 24

4.6 Cell-level policies 24

4.7 Filtered query results 24

4.8 George's cell-level diagnosis policies 25

4.9 Advantages of each filtering method 27

5.1 Average runtimes for the two filtering methods 31

List of Figures

3.1 The Trust Proxy 14

4.1 Alice's interaction with the open database using results filtering . . . 24
4.2 Alice's interaction with the database system using query modification 25
4.3 The demo query page 29
4.4 Query results 30

5.1 Average runtimes for the two filtering methods 32

Chapter 1 — Introduction

Concerns about the privacy of personal information are increasing significantly. Disclosing sensitive information to unauthorized entities often results in identity theft, financial loss, and other serious problems. This thesis focuses on the creation of a database system that is accessible to qualified strangers without the risks to privacy that such openness normally entails.

Hippocratic databases focus on protecting the privacy of the individuals whose personal information they contain. Requirements of Hippocratic databases [4] include the following:

Purpose Specification: Individuals whose personal information is stored in the database must be informed about the purposes for which their personal information will be used.

Consent: Personal information cannot be released for any purpose without the consent of the individuals to whom the information belongs.

Limited Disclosure: Information must not be disclosed for any other purpose.

Safety: Information stored in the database must be protected against theft and misuse.

Thus far, research in the area of Hippocratic databases [2, 13] has focused on managing policies and filtering data from queries. Little emphasis has been placed on authentication of users and verification of purposes. In addition, current databases are closed systems: only known, individually authorized users are allowed to access protected content.

This thesis presents an architecture for an open system that satisfies the requirements of a Hippocratic database [4] and leverages trust negotiation to authenticate users to roles. In this system, a role indicates the possession of a specific set of attributes, and roles replace the concept of purposes in Hippocratic databases. An open system does not authorize users by their *identity* (e.g., with a username and password); instead, users prove that they have required *attributes* by disclosing digital credentials. This allows individuals outside the local security domain to connect to the database and perform queries based on their membership in certain roles.

1.1 HIPAA and Privacy

While confidentiality is satisfied simply by not disclosing information to unauthorized individuals, privacy is more difficult to achieve. Privacy is defined as "freedom from unauthorized intrusion."[1] Privacy requires that information be protected even after its release to authorized individuals.

Protecting the privacy of personal information is becoming increasingly important, partly due to an increase in the availability of electronic information. In 2005, data from over 200,000 credit card accounts was stolen from CardSystems, a company that processes credit card transactions. CardSystems later admitted that it had been storing credit card information in violation of agreements it had with Visa and Mastercard [10]. Incidents such as this one illustrate the importance of keeping information safe.

In 1996, the United States Department of Health and Human Services introduced the Health Insurance Portability and Accountability Act (HIPAA). HIPAA was designed to increase efficiency in dealing with medical information and to manage the privacy of that information. HIPAA was passed into law and took effect in 2003. Today, entities who violate HIPAA can be fined up to $250,000 [20].

---

[1] Merriam-Webster Dictionary

The Privacy Rule is the component of HIPAA that deals with the privacy of medical information. It applies to electronic, written, and oral information [20]. Among other things, the Privacy Rule requires that patients be informed about who will have access to their medical information [21].

1.2 Database Access Control

Database systems provide access control at varying levels of granularity. It is common for database administrators to grant permission to access columns of a table, or to specify a filter that restricts access to certain rows [17]. A few recent systems (see Chapter 2) provide access control at the cell level.

Individuals' privacy requirements vary greatly. Some people are willing to disclose personal information to almost anyone. Others are adamant about disclosing information only when it is absolutely necessary. Cell-level access control allows database subjects to set their own privacy policies, opting in or out of certain disclosures. Cell-level policies allow the database to filter from query results only those cells that the query issuer does not have permission to view.

Database systems that provide cell-level access control are well-suited to HIPAA compliance because of their fine-grained nature. In order to provide a high level of privacy without filtering information unnecessarily, the system presented in this thesis provides access control at the cell level. The system utilizes two filtering methods that support cell-level access control, described in Sections 3.2.1 and 3.2.2.

1.3 Trust Negotiation

Many Internet transactions occur between strangers. Entities who interact on the Web should not disclose sensitive information to each other without first establishing a certain degree of trust. Parties engaged in such transactions can build trust by disclosing *digital credentials*, cryptographically signed statements that prove that their owners possess certain attributes.

Trust negotiation is an iterative process by which two parties disclose digital credentials in order to build trust in one another, based on the attributes they possess. For example, customers are more likely to trust an online bookstore that is a member of the Better Business Bureau (BBB). Trust negotiation makes use of access control policies, which protect sensitive resources and credentials. Policies specify the types of digital credentials (e.g., BBB credential) that the other party must disclose in order to gain access to resources. The parties use trust negotiation to prove that they satisfy the policies that protect each other's resources.

Individuals may consider their credentials to be sensitive resources and protect them with access control policies. Suppose that Fred, an AIDS Researcher, needs to access medical records from another hospital. The policy that protects these records requires Fred to be an AIDS Researcher and a Doctor. Fred has an AIDS Research credential, but he is unwilling to disclose it to entities that do not possess a Hospital credential. Once the hospital discloses its Hospital credential to Fred, he releases his AIDS Research credential. Trust negotiation often takes multiple rounds because credentials are sensitive. The two parties involved gradually gain trust in each other and exchange policies and credentials until the negotiation concludes.

1.4 Differences from Traditional Trust Negotiation

Traditional trust negotiation is coarse-grained. A negotiation-enabled server controls a set of resources and services, each with its corresponding access control policy. Managing this set of resources is simple: when a user requests a resource, the server looks up the policy associated with the resource, and it uses this policy in the ensuing negotiation. These negotiations use an all-or-nothing approach: if a user cannot satisfy all of the roles required in a policy, they are simply denied access to the resource.

In contrast, trust negotiation in database access control can be fine-grained and dynamic. Instead of requesting a single resource, query issuers essentially request many resources simultaneously (i.e., each item in the query results). A large number of different query results are possible, and each item in the query result might have a different access control policy. The result of fine-grained access control is that a partial negotiation failure does not imply denial of access to the entire query result.

1.5 Motivating Scenario

While vacationing in another part of the country, Alice is involved in an automobile accident. Bob, a physician in the emergency room at a nearby hospital, assesses Alice's condition. In order to provide Alice with the appropriate care, Bob needs information about her medical history. He connects to a medical database in Alice's home state that contains her information. Because Bob practices medicine in another state, he is unknown to the database and does not have a username and password to log in. Trust negotiation permits Bob to authenticate himself to the database by submitting his Doctor credential. This allows him to access the information he needs.

1.6 Goals

This thesis presents a database system that has the following characteristics:

1. **Open:** Allow qualified strangers to access the database by proving that they possess the required attributes.

2. **Fine-grained:** Provide cell-level access control.

3. **Customizable:** Allow individuals whose personal information is stored in the database to make choices regarding the disclosure of that information.

4. **Transparent:** Make the query filtering process as transparent as possible, so that users notice few differences between this system and a normal database.

1.7 Thesis Outline

The remainder of the thesis will proceed as follows. Chapter 2 discusses related work in database access control. Chapter 3 presents the design for an open database system and describes two filtering methods. Details about the implementation of the open database are given in Chapter 4. Chapters 5 and 6 contain a performance analysis and a threat model of the system, respectively. In Chapter 7, we discuss future work in this area and conclude.

Chapter 2 — Related Work

Mandatory Access Control (MAC) and Discretionary Access Control (DAC) are two common database security models [6, 8, 15]. Many current database systems build on one or both. In MAC, administrators assign security labels (e.g., Top Secret, Unclassified) to subjects (database users) and objects (data items). The database enforces access based on these labels. In DAC, data owners can grant permissions (e.g., read and write) to others. In some systems, owners can delegate the granting of permissions to other users [1, 24].

Many database systems support view-level access control [6]. A view contains a subset of columns and rows in a relation. Permissions are granted based on these views. Oracle supports the concept of a *virtual private database* [14], providing each user with their own "private" view of the tables based on certain filtering criteria.

Systems that support role-based access control (RBAC) [18] allow permissions to be granted to roles instead of to individual users. For example, an administrator might grant permission to view and update various tables to the Payroll group. When a new payroll employee is hired, they are assigned membership in the Payroll group and automatically inherit all of the privileges assigned to that group. In this way, RBAC greatly simplifies permission assignment. Usually, each registered database user belongs to a single role. Oracle supports role-based access control and allows users to act in more than one role at the same time [14].

Some database systems provide cell-level access control. Microsoft's SQL Server [16] supports both row-level and cell-level access control. Another cell-level system developed by LeFevre et al. [13] allows individuals to specify which fields of their personal information they are willing to disclose to whom. This is done using purpose-recipient pairs. For example, Bob might specify that he is willing to have his telephone number disclosed to charities for the purpose of solicitation, but not to reporters for any purpose. Before the results are returned to the query issuer, prohibited fields are replaced with *null* values.

Statistical databases attempt to provide statistical information about groups of individuals (e.g., averages and totals) without compromising the privacy of any individual. Beck demonstrates that any statistical database that provides statistically accurate data must necessarily leak information about individuals [5]. Beck's work shows that information can be leaked easily from databases in subtle and unexpected ways.

Database systems can be grouped into the following three broad categories, based on the type of access control they provide:

Freely accessible: Anyone can access the information in the database, without regard to their attributes or identity.

Closed: The system provides access only to a pre-established list of authorized users. No one else can access the database.

Open: The system provides access to users based on their attributes. Previously unknown individuals who can prove they are qualified can access the database.

Many freely accessible databases can be found on the Internet. For example, the Ancestral File[1] allows anyone to access a database of genealogical information and obtain birth and death dates, names of family members, and other data about individuals. This database can be accessed by strangers, but it does not restrict access to certain groups; rather, it allows disclosure of freely available information to any web user. Most traditional databases, such as Oracle and MySQL, are closed systems. The novelty of the database system presented in this thesis is that it is an open system, providing access based on attributes and not identity.

---

[1] http://www.familysearch.org

Chapter 3 — System Design

This chapter presents the design for an open database system. First, Section 3.1 describes the types of policies used to provide cell-level access control. Section 3.2 introduces the Trust Proxy, a software component that performs database filtering, and describes two filtering methods.

3.1 Policies

The system accomplishes cell-level access control through four different types of policies: column, row, cell, and content-based. This section describes each of these types of policies in detail.

3.1.1 Column Policies

A column policy governs the release of a column in the database. It is the default policy for a given type of information. For example, the column policy for telephone numbers might make them available to Doctors and Hospital Employees.

3.1.2 Row and Cell Policies

Individuals whose personal information is stored in the database (referred to as *database subjects* in this thesis) can make choices about the disclosure of their information through cell-level and row-level policies that they establish. Suppose that the diagnosis column policy makes diagnoses available to Doctors and Nurses. Sally, a patient, might opt to make her own diagnosis available only to Doctors. These personalized policies might apply to a single cell, as in the case of Sally's diagnosis, or to an entire row. When a cell policy exists, it supersedes the column policy for that cell. This is true even for cell policies that are more permissive than the corresponding column policy. Thus, database subjects can increase or decrease the privacy of their personal information.

A security expert, e.g., the database administrator, determines the privacy choices to offer to database subjects. In other words, subjects cannot write their own policies, but they are given a controlled set of choices about the disclosure of their personal information. This prevents database subjects from creating inappropriate policies to protect their data. For example, patients should not be allowed to create policies that prevent their doctor from obtaining their diagnosis information.

3.1.3 Content-Based Policies

Content-based access control is desirable in situations where access to data depends on the content of the data itself, as specified by constraints in column and cell policies.

Some content-based policies refer to attributes of the query issuer. For example, a content-based column policy for diagnoses might allow Nurses access only to diagnoses of patients on their assigned floor. Under this policy, diagnoses of patients on other floors are removed from the query results. Because access to diagnoses depends on the value of another column (floor), this is an example of an *outside-referencing* content-based policy.

Self-referencing content-based policies restrict access to a field based on the value of the field itself. For example, the diagnosis field might only be freely available when it is not “AIDS.” Then, for query issuers who have not been authorized to see “AIDS” diagnoses, those fields are replaced with *null*. Self-referencing content-based policies may leak sensitive information. For example, if *null* values are only used to mask diagnoses of terminal diseases, a query issuer can infer which patients have terminal diseases, even without authorization to see diagnosis information.

Since content-based policies can leak information, only outside-referencing content-based policies should be used. In addition, the column that the content-based policy references (e.g., floor in the example above) must not be sensitive. Otherwise, query issuers may be able to infer the content of the column based on any null values for the column in the query results.

Policies may themselves be considered sensitive resources. Seamons et al. [19] suggest ways to avoid disclosing policies inappropriately. Even without disclosing a policy, users may be able to make inferences about the values that were filtered by content-based policies. In order to avoid such inferences, policies should follow these minimum guidelines:

- Default policies should minimize disclosures.
- Self-referencing content-based policies should not be used.
- Some policies make strong implications about the data they protect. For instance, if a database subject has their diagnosis protected by the policy `AIDS Researcher`, it is likely that the patient's diagnosis is AIDS. Systems that allow policies to be disclosed to database users must not use these types of policies.

3.2 Trust Proxy

The Trust Proxy acts as an intermediary between a query issuer and the database (see Figure 3.1). The proxy performs filtering operations and negotiates trust with query issuers. The Trust Proxy is in the same administrative domain as the database, and in practice, the two may be located on the same machine.

This section presents two methods by which the Trust Proxy controls access to information in a database: 1) results filtering, and 2) query modification.

3.2.1 Results Filtering

Results filtering is an algorithm in which the Trust Proxy modifies query results in order to remove prohibited information. After the proxy verifies a query issuer's roles through trust negotiation, it evaluates the policies for the data in the user's query results and replaces field values that the user is not allowed to see with *null*.

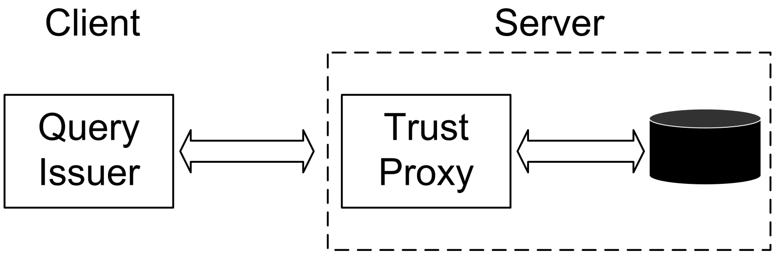


Figure 3.1: The Trust Proxy

The results filtering method can leak sensitive information. LeFevre et al. [13] point out that a query issuer can discover information such as a patient's diagnosis without being authorized to view that diagnosis. For example, suppose Mallory is a Nurse. She issues the query `SELECT name, diagnosis, phone FROM patients WHERE diagnosis = 'cancer.'` Mallory is authorized to view the 'name' column, since it is freely available, but she is only authorized to see diagnoses of patients who have opted to make their diagnoses available to Nurses. Each diagnosis she is not allowed to see is replaced with *null*. The query results are shown in Table 3.1. Even though some diagnoses have been filtered, Mallory knows that every patient listed in the query results has cancer.

For queries with a predicate (WHERE clause), such as the example above, the Trust Proxy must perform special filtering in order to avoid leaking information. Because the mere presence of Sally's and Reed's rows shows that they have cancer, those entire rows must be removed from the query results. More generally, whenever a query has a constraint on a particular column (e.g., `WHERE diagnosis = 'cancer'`), rows that are null in that column must be removed.

Rows that are null only in columns not referenced by the WHERE clause do not need to be removed. The row for Dan (Table 3.1) can remain in the query results without leaking information, since it only has a *null* value in the Phone column, which

Name	**Diagnosis**	**Phone**
Travis	cancer	555-7365
Sally	null	null
Reed	null	555-2329
Dan	cancer	null

Table 3.1: Query results for the query `SELECT name, diagnosis, phone FROM patients WHERE diagnosis = ‘cancer.’` Note the leaked diagnoses of Sally and Reed.

Name	**Diagnosis**	**Phone**
Travis	cancer	555-7365
Dan	cancer	null

Table 3.2: The modified query results

the WHERE clause does not reference. Table 3.2 shows the query results after the appropriate rows have been removed.

3.2.2 Query Modification

This section discusses query modification, the second filtering method. This method modifies queries in order to prevent the release of information to unauthorized individuals. Queries are modified based on inferred purpose-recipient pairs (e.g., charity for the purpose of solicitation), an algorithm introduced by LeFevre et al. [13], who do the modification. These pairs are determined from contextual information, not from the attributes of the query issuer, as is done in an open system

In an open system, the approach of LeFevre et al. can be adapted so that a user's authenticated roles replace purpose-recipient pairs. Roles are authenticated through trust negotiation and are used to rewrite queries so that the database only returns results for which the user is authorized. This removes the need for the Trust Proxy to filter results after they have been returned from the database.

The query modification filtering method requires query issuers to pre-select the roles in which they wish to act. Negotiation verifies that the query issuers belong to the roles that they pre-select. Once the Trust Proxy knows the roles to which the user has authenticated, it can modify the query. Details of the query modification algorithm are presented in Chapter 4.

Chapter 4 — Implementation

This chapter discusses details for an implementation of an open database system. Section 4.1 presents two options for storing policies. Sections 4.2 and 4.3 provide details about the results filtering and query modification algorithms. Section 4.4 shows how filtering occurs using the two methods in an example scenario. Section 4.5 compares the two filtering methods and explores the advantages of each. Finally, Section 4.6 describes a software demo constructed as part of this research.

4.1 Policy Storage

Policies are stored in the database. When a user issues a query, policies governing access to data in the query results are fetched from the database and used in the ensuing trust negotiation. This thesis presents two options for storing policies. First, actual role expressions can be stored as database policies. A role expression is simply a logical expression of roles that an individual must satisfy. We are not aware of any other database systems that use role expressions for policies. Using role expressions permits policies that involve a combination of roles. For instance, a role expression for diagnoses might be `((Nurse AND Researcher) OR Doctor)`. Table 4.1 shows how diagnosis policies are stored as role expressions.

Second, policies may be stored as separate permissions [13]. For a given information type (e.g., telephone number), a yes/no value is stored for each possible recipient type (e.g., Doctor). Database subjects can opt in or out of making their data available to the various groups, as shown in Table 4.2.

Storing policies as role expressions takes less space and is more flexible than storing them as separate permissions. For role expressions, there is one policy table for each data table, and the two tables have the same number of columns and rows. Each cell

Name	**Diagnosis**	**Diagnosis Policy**
John	Cancer	`Doctor OR Nurse`
Sally	Heart Attack	`Doctor`
Joe	Appendicitis	`Doctor OR Nurse OR Employee`

Table 4.1: Cell-level diagnosis policies stored as role expressions

Name	**Phone**	**Nurses?**	**Reporters**?
John	482-4458	Yes	Yes
Sally	257-8546	No	No
Joe	259-7445	Yes	No

Table 4.2: Cell-level telephone policies stored as separate permissions

---

in the data table has a corresponding cell in the policy table. Separate permissions require one policy table per column in the data table. For example, if the patients table has columns for name, diagnosis, and room number, three tables are needed to store name, diagnosis, and room number policies.

Role expressions require more processing time than separate permissions because they are logical expressions that need to be evaluated to determine whether the query issuer's roles satisfy the role expression. Separate permissions require no such evaluation.

4.2 Results Filtering Implementation

In this section, details of the results filtering algorithm are presented. In the results filtering method, when the user issues a database query, the Trust Proxy issues a separate query that uses the `SELECT DISTINCT` keywords to obtain a list of all the roles contained in access control policies for data referenced by the query. If the original query requested the name and room columns for all patients, the query used to obtain roles is

```
(SELECT DISTINCT namePolicy FROM patientsCellPolicies) UNION
(SELECT DISTINCT roomPolicy FROM patientsCellPolicies)
```

Next, the proxy builds a role expression that is a conjunction of all of the roles in the list. For example, if some information is protected by the policy `Doctor OR Nurse` and other information by `Hospital Employee`, the proxy combines these role expressions into a single composite role expression, which the proxy uses in the ensuing trust negotiation.

At this point, the user and the Trust Proxy begin trust negotiation. At the end of the negotiation, the Trust Proxy receives a negotiation response, which contains a list of roles for which the client (query issuer) was authenticated. Even if the negotiation failed (e.g., if the client could not satisfy all of the roles contained in the role expression), the query process can still continue. A failed role only means that some information may be filtered from the query results.

Next, the Trust Proxy augments the original query so that it also requests cell policies. If the original query was `SELECT name, diagnosis, room FROM patients`, the augmented query is

```
SELECT name, diagnosis, room, namePolicy, diagnosisPolicy, roomPolicy
   FROM patients, patientsCellPolicies
   WHERE patients.id = patientsCellPolicies.id.
```

This modification allows the proxy to fetch both data and cell policies in a single query, speeding up the filtering process. The Trust Proxy then requests column policies from the database. Once the proxy has the data and policies, it uses the list of authenticated roles to evaluate the role expressions for each cell in the query results in order to determine which cells the client is authorized to see. When a cell does not have a cell policy, the proxy uses the default (column) policy for that cell.

Cells whose policies evaluate to `false` are replaced with *null*. Rows that are *null* in all fields are removed from the query results. Any rows that are *null* in a column referenced by a WHERE clause are also removed. The proxy then returns the filtered query results to the query issuer.

4.3 Query Modification Implementation

This section describes the query modification method introduced by LeFevre et al. [13] and explains one way to extend it to an open system environment.

In one of their examples, patient telephone numbers are available to charities for the purpose of solicitation only for patients who have "opted in" to allow this disclosure. The original query, `SELECT Phone FROM Patients`, is rewritten to be:

```
SELECT CASE WHEN EXISTS
  (SELECT Phone_Choice FROM PatientChoices
   WHERE Patients.PatientID = PatientChoices.PatientID
   AND PatientChoices.Phone_Choice = 1)
THEN Phone ELSE null END FROM Patients
```

Here, `PatientChoices` is a table that records patients' preferences for having their data disclosed to charities for the purpose of solicitation. `Phone_Choice` is a column in this table that pertains to telephone numbers. Each row in the table with a value set to `1` in the `Phone_Choice` column indicates that a patient has opted to allow their telephone number to be disclosed to charities.

The SQL CASE statement is similar to the if-then-else statement used in many programming languages. In this example, the query checks cell policies for the phone column and only returns telephone numbers whose cell policies the query issuer satisfies. For individuals that have not made their telephone numbers available to charities,

Name	**Doctor**	**Nurse**	**Reporter**
John	Yes	Yes	Yes
Sally	Yes	No	No
Joe	Yes	Yes	No

Table 4.3: Cell-level policies for diagnoses

---

the inside SELECT statement returns an empty result set which, in turn, causes the CASE statement to evaluate to false. This causes a *null* value to replace the telephone numbers for those individuals.

In order to adapt the query modification method to an open system, cell policies are stored as separate permissions with one column for each recipient type, as described in Section 4.1. Each column's cell policies are stored in a separate relation. Table 4.3 shows example cell policies for diagnoses. As before, each row in the relation represents a patient. Each column represents a role. Each cell in the table indicates whether an individual's diagnosis is available to members of the role pertaining to the given column.

Consider the query `SELECT name, phone FROM patients`. Upon submitting her query, Alice checks a box that indicates she wants to act as a Nurse. Trust negotiation occurs before any query processing, and Alice is authenticated to the system as a Nurse. The Trust Proxy modifies her query so that it will only return cells that Nurses are authorized to view. The modified query is

```
SELECT name, CASE WHEN EXISTS
   (SELECT nurse
    FROM phoneCellPolicies
    WHERE patients.id = phoneCellPolicies.id
    AND phoneCellPolicies.nurse = ‘yes’)
THEN phone ELSE NULL END FROM patients
```

As before, columns referenced in a query's predicate (WHERE clause) must be handled differently. Suppose Alice changes her query to `SELECT name, phone FROM patients WHERE diagnosis = ‘cancer.’` In this case, the modified query is

```
SELECT name, CASE WHEN EXISTS
   (SELECT nurse
    FROM phoneCellPolicies
    WHERE patients.id = phoneCellPolicies.id
    AND phoneCellPolicies.nurse = ‘yes’)
THEN phone ELSE NULL END FROM patients WHERE EXISTS
   (SELECT nurse FROM diagnosisCellPolicies
    WHERE patients.id = diagnosisCellPolicies.id
    AND diagnosisCellPolicies.nurse = ‘yes’
    AND diagnosis = ‘cancer’)
```

The last five lines of this query ensure that when a patient's diagnosis is not available to Nurses, that entire row is excluded from the query results.

4.4 Example Scenario

Alice, a Nurse, connects to the hospital database and submits a query to obtain information about George, a new patient. The database access control policy specifies that names are freely available. Diagnoses are only disclosed to Doctors and Nurses. These column policies are shown in Table 4.4. George, however, has opted not to allow Nurses to see his diagnosis.

This section shows how the Trust Proxy performs filtering for Alice's query. Section 4.4.1 illustrates the process using results filtering. Section 4.4.2 shows how query

Column	**Policy**
Name	Freely available
Diagnosis	`Doctor OR Nurse`
Room	Freely available
Telephone	`Employee`

Table 4.4: Column Policies

modification filters the same query.

4.4.1 Results Filtering

Alice issues a normal database query to find information about George. Her query is `SELECT * FROM patients WHERE name = 'George'`. The filtering process is shown in Figure 4.1. The Trust Proxy receives the query and issues a separate query to the database requesting roles for the user's query. The Trust Proxy uses the list of roles it receives to create a role expression, then initiates trust negotiation with Alice.

During the negotiation, Alice submits her Nurse credential and thus authenticates to that role. At this point, the proxy sends Alice's original query to the database and also requests the policies that protect Alice's query results. The database then returns the full query results to the proxy, along with the associated policies. The unfiltered query results are shown in Table 4.5. The proxy uses the list of authenticated roles to filter the query results. Because George's diagnosis is not available to Nurses, the proxy replaces it with *null*. The proxy then sends the query results on to Alice. Filtered query results are shown in Table 4.7.

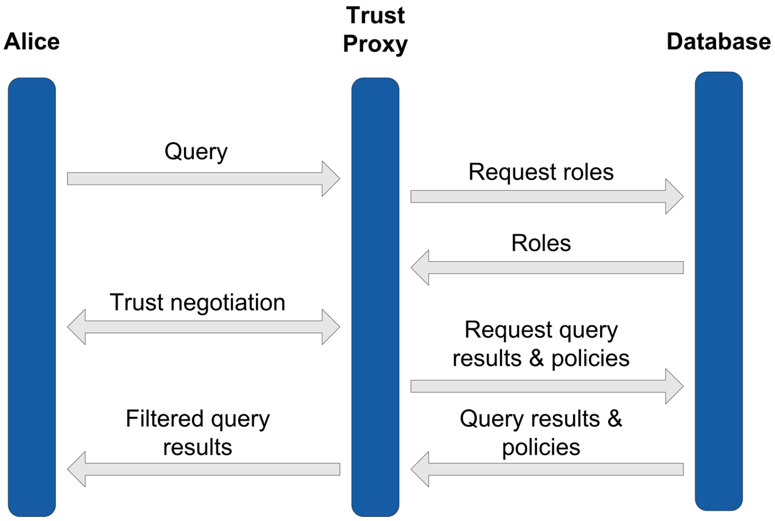

Alice
Trust Proxy
Database
Query
Request roles
Roles
Trust negotiation
Request query results & policies
Filtered query results
Query results & policies

Figure 4.1: Alice's interaction with the open database using results filtering

Patient ID	**Name**	**Diagnosis**	**Telephone**
1234567	George	Emphysema	555-1725

Table 4.5: Unfiltered query results

Patient ID	**Name**	**Diagnosis**	**Telephone**
1234567	freely available	Doctor	Doctor or Nurse

Table 4.6: Cell-level policies

Patient ID	**Name**	**Diagnosis**	**Telephone**
1234567	George	null	555-1725

Table 4.7: Filtered query results

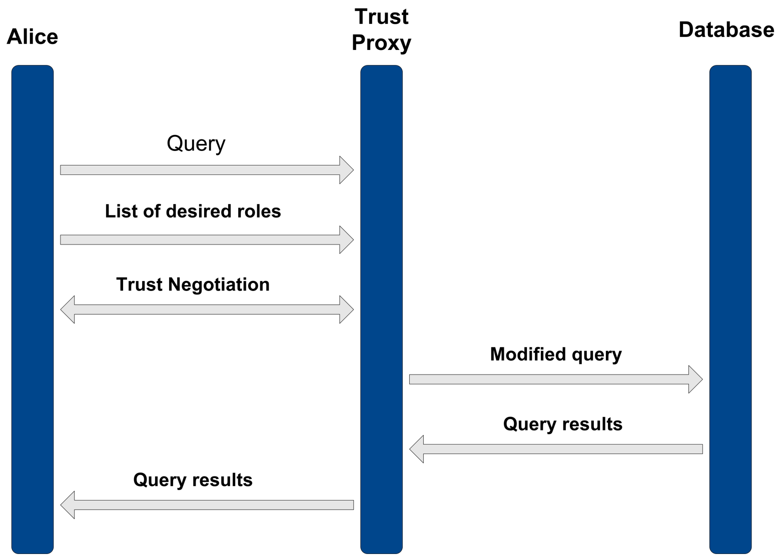

Alice
Trust Proxy
Database
Query
List of desired roles
Trust Negotiation
Modified query
Query results
Query results

Figure 4.2: Alice's interaction with the database system using query modification

Patient ID	Doctor?	Nurse?	Employee?
1234567	yes	no	no

Table 4.8: George's cell-level diagnosis policies

4.4.2 Query Modification

Figure 4.2 shows Alice's interaction with the database using query modification. When she submits the query, Alice notifies the proxy that she wants to act in the role of a Nurse for this query. The proxy initiates a trust negotiation with Alice and verifies that she satisfies the role `Nurse`. The proxy uses that role to modify Alice's query so that it will only return results that Nurses are authorized to see. Table 4.8 shows George's cell-level policies for his diagnosis, which indicate that it is not available to Nurses. His policy for his telephone number, however, does permit access to Nurses. This query returns the same results as those shown in Table 4.7.

4.5 Comparison of Filtering Methods

This section discusses strengths and weaknesses of the two filtering methods presented in this thesis.

Results filtering supports more flexible policies than query modification. Since query modification requires policies to be stored as separate questions (a ‘yes’ or ‘no’ for each recipient type), policies are a simple disjunction of roles. For example, if Bob has made his telephone number available to Doctors and Nurses, his policy is `Doctor OR Nurse`. More complex role expressions, e.g., `Doctor OR (Nurse AND Researcher)` are not supported. Results filtering, on the other hand, stores policies as role expressions and thus supports complex policies such as the one used above.

Forcing a user to pre-select roles for the query modification method has subtle privacy implications. For instance, suppose Bob unknowingly connects to a malicious server and selects the role `AIDS Researcher` before submitting a query. The server can infer that Bob is an AIDS Researcher without authenticating itself to Bob. Bob may be unwilling to disclose to untrusted servers that he is a member of the AIDS Research group, but he allows a server to infer that information each time he preselects the AIDS role.

Query modification is, by nature, faster and simpler than results filtering. Results filtering requires that the Trust Proxy evaluate query results cell by cell. In contrast, query modification leverages the capabilities of the query language to filter the results in the database during query processing. Chapter 5 compares the performance of the two filtering methods.

Results filtering and query modification handle policies differently, and this can affect the information that query issuers can infer following a negotiation. Suppose Alice submits a query about John, who has his diagnosis protected by the policy `AIDS Researcher`. This query returns a result set with a single row. If filtering

Results Filtering	**Query Modification**
Allows complex policies	Faster and simpler
No need to pre-select roles	Policies not disclosed to query issuer

Table 4.9: Advantages of each filtering method

happens using query modification, Alice simply pre-selects her role (`Nurse`), and she is never notified why John's diagnosis was replaced with *null*. However, if results filtering is used, the Trust Proxy creates a role expression for the result set, `Doctor AND Nurse AND AIDS Researcher`, and this role expression is available to Alice. The AIDS Researcher role in the role expression tells Alice that one or more cells in the query results are protected by the role `AIDS Researcher`, which is a good indication that John's diagnosis is AIDS.

A summary of the strengths of each filtering method is shown in Table 4.9.

4.6 Software

Our prototype software is written in Java and implements both results filtering and query modification. It interacts with MySQL databases, but changing the software to connect to another database system would be trivial.

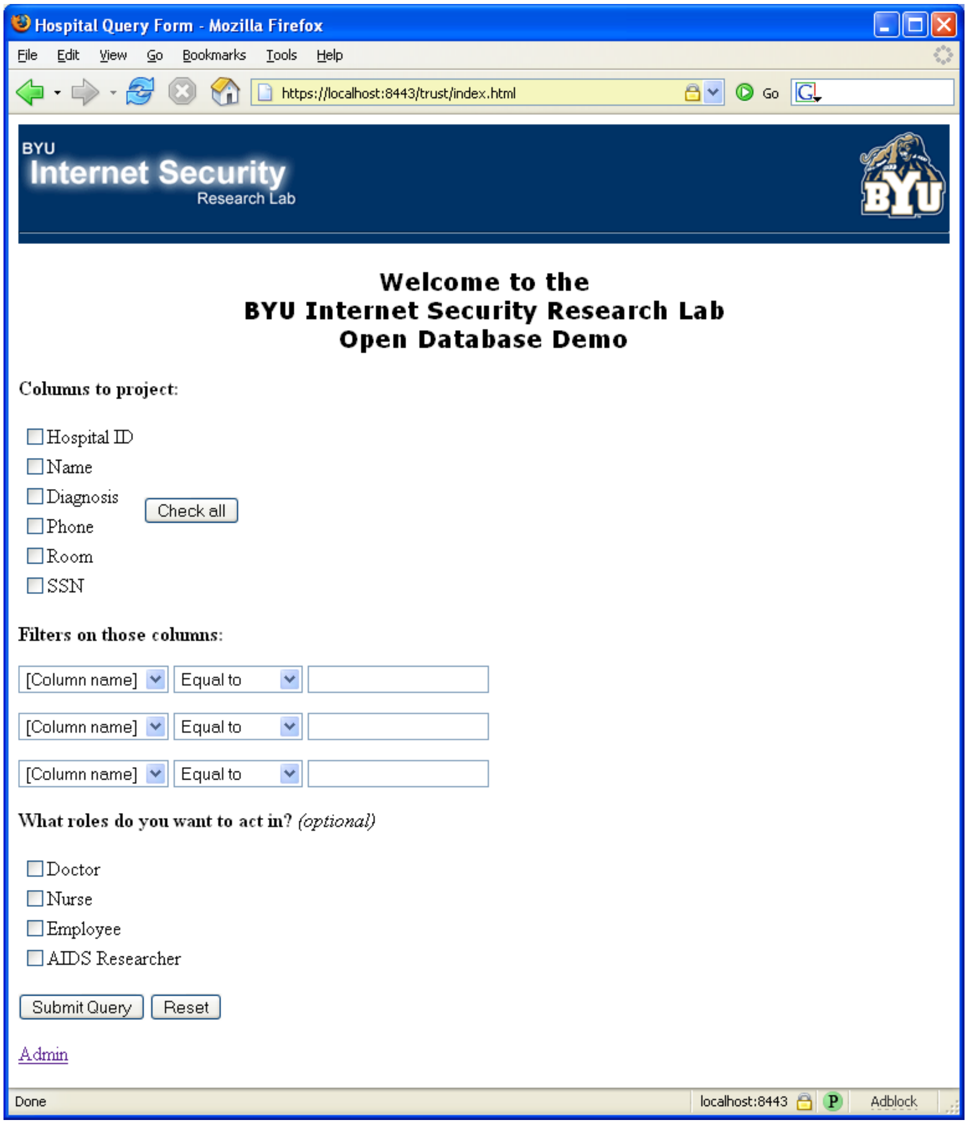
TrustBuilder is a prototype trust negotiation system developed by researchers in the Internet Security Research Lab. It supports X.509v3 certificates. Users prove membership in roles by releasing the appropriate credentials to the other party involved in the negotiation.

When a TrustBuilder negotiation terminates, the server sends a negotiation response object to the client indicating whether the negotiation succeeded or failed. Since trust negotiation previously used an all-or-nothing approach, the response did not notify the client which roles were authenticated during the negotiation. In order to accommodate fine-grained access control, the negotiation response object was extended to include a list of authenticated roles. The Trust Proxy uses this list to filter information that is only available to roles to which the client did not authenticate.

The implementation combines two types of policies: database policies and Trust Policy Language (TPL) policies [12]. Database policies have been discussed throughout this thesis. TPL policies describe the credentials a user must disclose in order to authenticate to the roles in the database policies. For example, a database policy might make hospital room numbers available to the `Nurse` role. A TPL policy will specify that the `Nurse` role requires the user to submit either a Registered Nurse credential or a Licensed Practical Nurse credential, signed by the American Nursing Association.

The proof-of-concept software demonstration provided with this thesis uses Tomcat and Java Servlets. The demo allows the user to assume the role of one of four fictional health care workers, each possessing different credentials. The user specifies a query by filling in an HTML form that has check boxes for the available columns in the database and text fields to enter conditions (e.g., `‘WHERE room = 205’`). There are also check boxes that allow the user to specify the roles to assume during the current query. If the user checks any of the boxes, the query modification method is used. Otherwise, results filtering is used. A screenshot of the query page is shown in Figure 4.3.

The Trust Proxy intercepts queries, and trust negotiation and filtering proceed as described in Sections 3.2.1 and 3.2.2. Query results are presented to the user in an HTML table, as shown in Figure 4.4.


Hospital Query Form - Mozilla Firefox
File
Edit
View
Go
Bookmarks
Tools
Help
https://localhost:8443/trust/index.html
Go
BYU
Internet Security
Research Lab
BYU
Done
localhost:8443
P
Adblock

Welcome to the
BYU Internet Security Research Lab
Open Database Demo

Columns to project:

☐ Hospital ID
☐ Name
☐ Diagnosis
☐ Phone
☐ Room
☐ SSN

Check all

Filters on those columns:

[Column name] Equal to
[Column name] Equal to
[Column name] Equal to

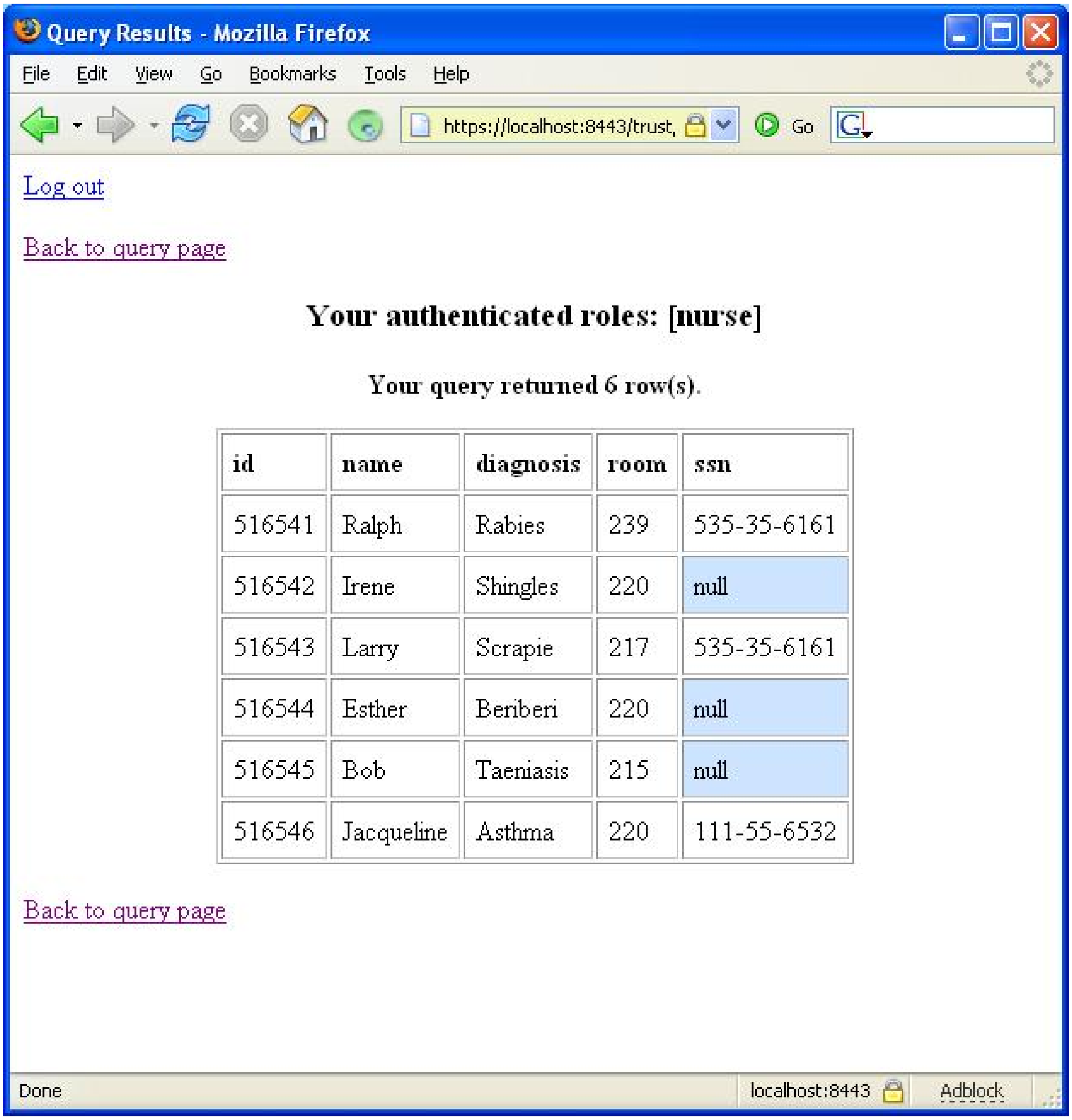
What roles do you want to act in? *(optional)*

☐ Doctor
☐ Nurse
☐ Employee
☐ AIDS Researcher

Submit Query Reset

Admin

Figure 4.3: The demo query page


Query Results - Mozilla Firefox
File
Edit
View
Go
Bookmarks
Tools
Help
https://localhost:8443/trust,
Go
Done
localhost:8443
Adblock

Log out

Back to query page

Your authenticated roles: [nurse]

Your query returned 6 row(s).

id	name	diagnosis	room	ssn
516541	Ralph	Rabies	239	535-35-6161
516542	Irene	Shingles	220	null
516543	Larry	Scrapie	217	535-35-6161
516544	Esther	Beriberi	220	null
516545	Bob	Taeniasis	215	null
516546	Jacqueline	Asthma	220	111-55-6532

Back to query page

Figure 4.4: Query results

Chapter 5 — Performance Analysis

This chapter examines the runtime performance of the open database system prototype. Specifically, this chapter compares the filtering times of the two methods for result sets of varying sizes. The performance results in this chapter measure only the times for database filtering operations. They exclude times for trust negotiation. This permits a more accurate analysis and comparison of the runtimes for the implementations of the algorithms described in this thesis.

The tests were performed on a 3.4 GHz Pentium 4 machine running Windows XP Professional with 1 GB of RAM. The software was tested in this configuration on query result sets ranging from 10 to 10,000 rows. Runtimes for the query `SELECT * FROM patients` are shown in Table 5.1, and the corresponding graph is shown in Figure 5.1. Each result is an average of ten execution times for a particular result size. Note that each of the times includes a constant startup time of slightly less than 300 ms, most of which is spent creating a connection to the database.

For relatively small result sets (i.e., a few thousand rows or less), results filtering is slightly faster than query modification. This is probably due to the extremely long

Rows	**Results Filtering**	**Query Modification**
10	333 ms	375 ms
100	325 ms	411 ms
1000	388 ms	472 ms
10000	942 ms	908 ms

Table 5.1: Average runtimes for the two filtering methods

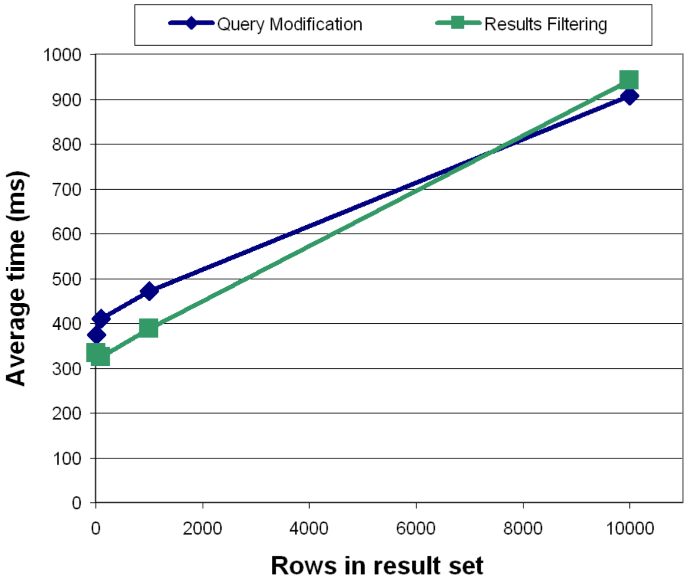


Figure 5.1: Average runtimes for the two filtering methods

and complex queries that are produced by the query modification algorithm. The query is just as complex for the result set of 10 rows as for the larger result sets, and database processing takes longer for these queries than for the much simpler queries used by results filtering. As result sets grow larger, query modification becomes the faster of the two algorithms. For both methods, filtering always completes in less than a second for the tests performed in this analysis.

Typical performance for a simple negotiation using TrustBuilder is about two seconds. When a user executes more than one query in a single database session, performing trust negotiation each time to authenticate a user to the same roles is wasteful. For this reason, the Trust Proxy caches roles for the duration of a user's session. If Bob submits a query and authenticates himself as a Doctor through trust negotiation, then submits another query where he needs to assume the same role, trust negotiation does not occur the second time. However, if Bob's second query requires him to act in a role he did not need for the first query (e.g., the second query requires him to be an AIDS Researcher), trust negotiation occurs the second time but only for that new role. There is no need to reauthenticate Bob as a Doctor.

Chapter 6 — Threat Model

This chapter discusses threats to the open database system and explores methods to resolve them.

Each time a query is issued, potentially sensitive information is sent between the Trust Proxy and the database, and between the Trust Proxy and the user. Using Transport Layer Security (TLS) ensures the confidentiality and integrity of the information in transit.

The Trust Proxy can be a single point of failure for both filtering methods. It is very important to secure the Trust Proxy against all known types of security threats. However, protecting the Trust Proxy is no different from protecting any other server on the Internet.

It is also important to prevent unauthorized changes to policies (willful or accidental). Care should be taken to ensure that a minimum number of individuals have write access to database policy tables. Database administrators are an example of those who should have this access. When individuals want to change policies that protect their information, they submit a request to the administrators, who authenticate to the database and perform the actual updates.

A common fear in database access control is that a malicious user might use a combination of queries to make inferences about information they are not allowed to see. For instance, suppose Tom, a hospital employee, is aware of the existence of Bob's information in a medical database and suspects that Bob's diagnosis is 'cancer'. He issues the query `SELECT * FROM patients WHERE diagnosis = ‘cancer.’` A row for Bob is not returned in the query results. Tom does not know if this is because Bob's diagnosis was 'cancer' or because Bob opted out of allowing employees to see his diagnosis. As explained in Section 3.2.1, the entire row for Bob was removed because it was null in the diagnosis column, which was referenced by the WHERE clauses. Tom decides to submit more queries in order to deduce Bob's diagnosis. He submits the query `SELECT * FROM patients WHERE diagnosis != ‘cancer.’` Bob's diagnosis does not appear in this result set either. In fact, rows for other patients who have opted out of letting employees see their diagnosis do not appear in either result set. Thus, with these two queries, the only information Tom can infer is that he is not authorized to see Bob's diagnosis.

Next, Tom issues queries to find patients that have other common diseases. This still does not provide him with any information about Bob's diagnosis. Even if Tom has a comprehensive list of all diseases in the database, he still cannot obtain any information about Bob's diagnosis through a process of elimination. Bob's row (and probably rows for other patients) is not returned in the result set for any diagnosis; again, this is not because it is ‘cancer,’ but because Tom is not allowed to see it. Finally, Tom removes the WHERE clause and submits the simple query `SELECT * FROM patients.` Here, Bob's row finally appears in the query results, but Bob's diagnosis has been replaced with *null.* Again, this only tells Tom that he is not authorized to view Bob's diagnosis, but he still knows nothing about the diagnosis.

Database joins combine data from two or more tables into a single result set. In this open system, query issuers cannot use joins to obtain information for which they are not authorized because joins use a WHERE clause. For example, the query

```
SELECT name, medication FROM patients, diagnosisMedications
   WHERE patients.diagnosis = ’aids’ and patients.diagnosis =
   diagnosisMedications.diagnosis
```

lists medications typically prescribed for patients who have AIDS. A query issuer cannot use the returned medications to infer a patient's diagnosis because rows where the diagnosis is *null* are removed due to the WHERE clause's reference to that column (see Section 3.2.1). As a result, query issuers only see the names and medications for patients whose diagnoses they have permission to view.

Chapter 7 — Conclusions and Future Work

This thesis presented the design and implementation of an open system database. Whereas traditional databases rely on a username/password access control model, the system authenticates users based on their attributes through trust negotiation. This system allows qualified strangers to access the database without compromising the privacy of the information stored in the database.

This thesis described and analyzed two methods for access control in open Hippocratic databases. This thesis adapted a query modification method proposed by LeFevre et al. [13] for use in an open system. Instead of using query modification with inferred purpose-recipient pairs, queries are modified based on a user's roles, which are authenticated through trust negotiation.

We also proposed and implemented a novel approach to database access control that uses a combination of column and cell-level policies. This method overcomes some limitations of the query modification method by supporting policies that use a combination of roles and not requiring query issuers to choose their roles before negotiation takes place.

Cell-level access control maximizes the amount of information the database can disclose to query issuers without violating the policies that protect the data, increasing the usefulness of the system. Giving database subjects the ability to tailor policies gives them control over their own privacy.

This research focused on flexibility. It enables qualified strangers to access the database and receive as much information as possible. Performance was a secondary concern. While added security almost always has a negative effect on performance, it is important to ensure that the system remains reasonably fast. Performance testing shows that both query modification and results filtering complete within a reasonable amount of time in our prototype system using a sample database of 10,000 records.

Many aspects of this research area remain unexplored. For example, this thesis focused on assigning read permissions. The design presented in this thesis could be augmented to include insert, update, and delete permissions. Policies for these operations could be stored and handled in a similar manner to read permissions.

Encryption is a particularly powerful feature provided by some databases. Encryption in a database has the significant advantage that even if unauthorized individuals gain access to sensitive information, it will be in a form that they cannot read without knowledge of the encryption key.

Integrating the features described in this thesis directly into an open source database like MySQL would allow filtering to occur in the database system itself, eliminating the need for a Trust Proxy.

References

[1] Mysql reference manual. http://downloads.mysql.com/docs/refman-5.1-en.pdf, Mar 2006.

[2] Rakesh Agrawal, Dmitri Asonov, Roberto Bayardo, Tyrone Grandison, Christopher Johnson, and Jerry Kiernan. White paper: Managing disclosure of private health data with hippocratic databases. http://www.almaden.ibm.com/software/projects/hdb/Publications/papers/nc_hdb_white_paper_health.pdf, Jan 2005.

[3] Rakesh Agrawal, Paul Bird, Tyrone Grandison, Jerry Kiernan, Scott Logan, and Walid Rjaibi. Extending relational database systems to automatically enforce privacy policies. In *Proceedings of the 21st International Conference on Data Engineering*, Tokyo, Japan, Apr 2005. IEEE Computer Society.

[4] Rakesh Agrawal, Jerry Kiernan, Ramakrishnan Srikant, and Yirong Xu. Hippocratic databases. In *Proceedings of the 28th VLDB Conference*, Hong Kong, China, Aug 2002.

[5] Leland L. Beck. A security mechanism for statistical databases. *ACM Transactions on Database Systems*, 5(3), 1980.

[6] Elisa Bertino and Ravi Sandhu. Database security–concepts, approaches, and challenges. *IEEE Transactions on Dependable and Secure Computing*, 2(1), 2005.

[7] Robert Bradshaw, Jason Holt, and Kent E. Seamons. Concealing complex policies with hidden credentials. In *Proceedings of the Eleventh ACM Conference on Computer and Communications Security*, Washington, DC, Oct 2004.

[8] Silvana Castano, Mariagrazia Fugini, Giancarlo Martella, and Pierangela Samarati. *Database Security*. ACM Press, 1995.

[9] Microsoft Corporation. Cell-level security in SQL Server 7.0 OLAP services. http://www.microsoft.com/technet/prodtechnol/sql/70/maintain/cellsec.mspx, 2001.

[10] Julie Creswell and Eric Dash. Banks unsure which cards were exposed in breach. http://www.nytimes.com/2005/06/21/business/21card.html, 2005.

[11] Luigi Giuri and Pietro Iglio. Role templates for content-based access control. In *Proceedings of the Second ACM Workshop on Role-based Access Control*, Fairfax, Virginia, Nov 1997. ACM Press.

[12] Amir Herzberg, Yosi Mass, Joris Michaeli, Dalit Naor, and Yiftach Ravid. Access control meets public key infrastructure, or: Assigning roles to strangers. In *Proceedings of the 2000 IEEE Symposium on Security and Privacy*, Oakland, California, May 2000.

[13] Kristen LeFevre, Rakesh Agrawal, Vuk Ercegovac, Raghu Ramakrishnan, Yirong Xu, and David DeWitt. Limiting disclosure in hippocratic databases. In *Proceedings of the 30th VLDB Conference*, Toronto, Canada, Aug 2004.

[14] Arup Nanda and Donald K. Burleson. *Oracle Privacy Security Auditing*. Rampant, 2003.

[15] Sylvia Osborn, Ravi Sandhu, and Qamar Munawer. Configuring role-based access control to enforce mandatory and discretionary access control policies. *ACM Transactions on Information and System Security*, 3(2), 2000.

[16] Art Rask, Don Rubin, and Bill Neumann. Implementing row- and cell-level security in classified databases using SQL Server 2005. http://www.microsoft.com/technet/prodtechnol/sql/2005/multisec.mspx, Apr 2005.

[17] Shariq Rizvi, Alberto Mendelzon, S. Sudarshan, and Prasan Roy. Extending query rewriting techniques for fine-grained access control. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, Paris, France, Jun 2004.

[18] Ravi Sandhu, David Ferraiolo, and Richard Kuhn. The NIST model for role-based access control: Towards a unified standard. In *Proceedings of the 5th ACM Workshop on Role Based Access Control*, Berlin, Germany, Jul 2000.

[19] Kent E. Seamons, Marianne Winslett, and Ting Yu. Limiting the disclosure of access control policies during automated trust negotiation. In *Proceedings of the Network and Distributed System Security Symposium*, San Diego, California, Feb 2001.

[20] Phoenix Health Systems. What's HIPAA? a basic HIPAA primer. http://www.hipaadvisory.com/regs/HIPAAprimer.htm.

[21] US Department of Health and Human Services. Standards for privacy of individually identifiable health information. http://www.hhs.gov/ocr/hipaa/privrulepd.pdf, 2002.

[22] William H. Winsborough, Kent E. Seamons, and Vicki E. Jones. Automated trust negotiation. In *Proceedings of the DARPA Information Survivability Conference and Exposition*, Hilton Head, South Carolina, Jan 2000.

[23] Ting Yu. Fine-grained database access control. In *Proceedings of the Fall Privacy Place Workshop*, Raleigh, North Carolina, Oct 2004.

[24] Longhua Zhang, Gail-Joon Ahn, and Bei-Tseng Chu. A role-based delegation framework for healthcare information systems. In *Proceedings of the 7th ACM Symposium on Access Control Models and Technologies*, Monterey, California, Jun 2002.