**CS448 - Project 3**

**Role-based Access Control for Cloud Data**

Credit: 5 points

Due date: 04/23/2014

**Project Description:**

In this project, you will implement a simple role-based access control scheme for data in a private cloud. The project has three goals:

1. Using the Amazon Relational Database Service (RDS) to obtain data from a cloud service
2. Building a simple role-based access control scheme.
3. Building a simple data encryption mechanism to hide data from database users with insufficient access privileges.

**Preliminaries:**

1. **Role-based Access Control (RBAC):**

Role-based access control (RBAC) is a popular data protection mechanism, where the operations a user can perform on specific data is determined by his data access roles. In this scheme, data access rights are granted to groups having specific roles instead of individual users. Users can have multiple roles and are entitled to all privileges of the roles they have.

In this project, we use a simple RBAC mechanism, where regular users (non-admin) of the database are assigned roles by the administrator, who also assigns specific permissions regarding the use of the database tables to each role. For simplicity, the only operations regular users of the database can perform on the data are SELECT and INSERT. Owner of a data item can restrict access to the item by making it available in plaintext only to users with a specific role, and hiding some fields of the data from other users using encryption. Different from subschemas, a user can see all fields of a table if he has the privilege for the SELECT operation on that table, but may see encrypted versions of some of fields of some data items based on his role’s permissions for that item.

1. **Data Encryption using the Shift (Caesar) Cipher:**

A shift cipher is a very simple data encryption mechanism, where each character in a word is substituted with the same character shifted by a specific number of characters to produce the corresponding cipher-text (i.e. encrypted text). The encryption/decryption key for this kind of cipher is the number of characters to shift by. For example, given the plaintext *Alice* and the encryption key *2*, you calculate the corresponding cipher-text as follows:

First character: A, A shifted right by 2: C

Second character: l, l shifted right by 2: n

Third character: i, i shifted right by 2: k

Fourth character: c, c shifted right by 2: e

Fifth character: e, e shifted right by 2: g

Therefore the cipher-text for the plaintext *Alice* is *Cnkeg* when the encryption key is 2*.*

You can implement this type of cipher in Java by converting the plaintext string to a char array and adding the encryption key value to each character one-by-one for encryption. For decryption of a cipher-text to get the original text (plaintext), you perform the exact opposite operation, i.e. subtract the encryption key value from each character in the cipher-text. This is possible because each character has an integer value.

For this project, you can assume that we will not test boundary conditions (such as the encrypted character value going beyond the letter z). Also, the strings to be encrypted will only consist of letters.

1. **Connecting to Amazon RDS:**

The dataset you will use in this project resides in an Oracle database on Amazon’s Relational Data Services (RDS) cloud. Your program should connect to this database using a database connection library (JDBC). To get started using JDBC, you can go through the tutorial at <http://www.tutorialspoint.com/jdbc/jdbc-quick-guide.htm>. You can also see other sample code at <https://www.cs.purdue.edu/homes/bb/cs448/StudentDB.pdf>. These two resources should provide sufficient knowledge about JDBC for this project. Note that when registering the database driver in your code, you should pass the argument “oracle.jdbc.driver.OracleDriver” to the Class.forName() method, as you will be connecting to an Oracle database. Also, you will need to include the JDBC driver [**ojdbc6.jar**](http://cs.purdue.edu/homes/bb/cs448/ojdbc6.jar) in your classpath at runtime to be able to use the database connectivity functions.

You should use the following information to connect to the database on RDS:

Connection string (on a single line, with no spaces):

jdbc:oracle:thin:@(DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(HOST=cs448db.cdo5tavfgia0.us-east-1.rds.amazonaws.com)(PORT=1521))(CONNECT\_DATA=(SID=irisdb)))

username: your career account username

password: your career account **username**

You should use the above parameter values with the *getConnection* method of the *DriverManager* class when connecting to the database.

**Database Schema**

The database in this project consists of the tables listed below. Note that these tables are already available in the database, i.e. you do not need to create them.

**Admin tables:**

These tables are for administrative tasks only and are not accessible by regular users.

USERS(username:string, password:string)

ROLES(roleName:string, encryptionKey:integer)

USERROLES(username:string, roleName:string)

ROLEPRIVILEGES(roleName:string, tableName:string, privName:string)

**User tables:**

There will only be three tables accessible by regular users: DEPARTMENT, STUDENT and FACULTY. The users will still need to have the appropriate permissions to view data in/insert data into these tables. Note that each of these tables has the fields *owner* and *encryptedColumn,* the functions of which are explained below. These two columns should **not** be output when a SELECT command is issued by any user.

DEPARTMENT(deptid:string, dname:string, location:string, owner:string, encryptedColumn:integer)

STUDENT(snum:string, sname:string, deptid:string, slevel:string, age:string, owner:string, encryptedColumn:integer)

FACULTY(fid:string, fname:string, deptid:string, owner:string, encryptedColumn:integer)

**Command Set**

Your program should accept input from the command line (with each command terminated with a newline character), process the input and output a response until the user quits the program. The program should implement the command set listed below. Note that the reserved words are in boldface and case-sensitive. There will be a single space between each word and no spaces between values in an attribute value list (i.e. no spaces before and after parentheses or commas). Note that the commands will always be input in the correct format, i.e. you do not need to check for syntax errors.

**LOGIN** *username* *password*

Upon issuance of this command, you should check that the provided username-password pair matches the one in the USERS table in the database. If not, you should print the error message “Invalid login”. If a match is found, you should update the current user to the one specified by *username* and print “Login successful”*.*

**CREATE ROLE** *roleName encKey*

If the current user is the admin when this command is issued, you should insert a new record in the ROLES table with the primary key *roleName* and encryptionKey *encKey.* You do not need to check if this role already exists in the database. If the current user is not the admin, you should print the error message “Authorization failure”.

**CREATE USER** *username password*

If the current user is the admin when this command is issued, you should insert a new record in the USERS table with the attribute values (*username, password*) and print “User created successfully”. You do not need to check if this user already exists in the database. If the current user is not the admin, you should print the error message “Authorization failure”.

**GRANT ROLE** *username* *roleName*

If the current user is the admin when this command is issued, you should insert a new record in the USERROLES table with the attribute values *(username, roleName)* and print “Role assigned successfully”. You do not need to check if this user exists in the USERS table, roleName exists in the ROLES table or this pair already exists in the USERROLES table. If the current user is not the admin, you should print the error message “Authorization failure”.

**GRANT PRIVILEGE** *privName* **TO** *roleName* **ON** *tableName*

If the current user is the admin when this command is issued, you should insert a new record in the ROLEPRIVILEGES table with the attribute values *(roleName, tableName, privName)*. You do not need to check whether this privilege already exists in the database. If the current user is not the admin, you should print the error message “Authorization failure”.

**REVOKE PRIVILEGE** *privName* **FROM** *roleName* **ON** *tableName*

If the current user is the admin when this command is issued, you should delete the record in the ROLEPRIVILEGES table with the attribute values *(roleName, tableName, privName)*. You do not need to check whether this privilege already exists in the database. If the current user is not the admin, you should print the error message “Authorization failure”.

**INSERT INTO** *tableName* **VALUES(***valueList***)** **ENCRYPT** *columnNo ownerRole*

Upon issuance of this command, you should first check if any of the roles of the current user has the “insert” privilege on the table *tableName*. If not, you should print the error message “Authorization failure”. Otherwise, you should insert the values in *valueList* (which is a list of comma separated strings **none** of which are enclosed in quotes) into the table *tableName*. While inserting the tuple, you should set the *owner* attribute value to *ownerRole* and set the value of the *encryptedColumn* attribute to *columnNo*. You do not need to check whether this record already exists in the database. Before insertion, you should check the value of *columnNo*: If it is greater than 0, you need to encrypt the corresponding attribute value with the encryption key of the owner role (which is found in the ROLES table). *columnNo* specifies the order of the column to be encrypted in the table and will never be negative or greater than *number of columns in the table -2*. For example, if *columnNo* is 2 when inserting into the table DEPARTMENT, the value to be inserted for the column *dname* should be encrypted.

**SELECT** **\* FROM** *tableName*

Upon issuance of this command, you should first check if any of the roles of the current user has the “select” privilege on the table *tableName*. If not, you should print the error message “Authorization failure”. Otherwise, you should print the names of the attributes in this table on a line by itself, each separated by a comma and all records in the table *tableName*, one record per line, with each attribute value separated by a comma (attribute values in the same order as the attribute names listed on the first line). The value printed for each attribute will depend on the owner of the data. If any of the current user’s roles is the owner role of a tuple, all attribute values for that tuple should be printed in plain text (i.e. encrypted attribute values should be decrypted before printing), otherwise attribute values should be printed as they stored in the database. You can assume that there will be at least one record in the table *tableName* when this command is issued.

**QUIT**

Your program should terminate upon issuance of this command.

**Submission Instructions:**

The project should be implemented in Java and your main file should be named RBAC.java.

Please create a README file containing identifying information. For example:

CS448 - Project 3

Author: John Doe

Login: jdoe

Email: jdoe@cs.purdue.edu

Include here anything you might want us to know when grading your project.

To turn in your project, ssh to lore.cs.purdue.edu, create a folder named project3 in your home directory and copy RBAC.java and your README file to that folder.

After copying your files in the folder project3, execute the following command in your home directory:

turnin –c cs448 –p proj3 project3

To verify the contents of your submission, execute the following command right after submission:

turnin –c cs448 –p proj3 -v