```java
package ballmerpeak.turtlenet.server;

import ballmerpeak.turtlenet.shared.Message;
import ballmerpeak.turtlenet.shared.Conversation;
import java.security.*;
import java.sql.*;
import java.security.*;
import java.util.List;
import java.io.File;
import java.util.Vector;
import java.util.Arrays;

public class Database {
    public static String path = "./db"; //path to database directory
    private Connection dbConnection;
    private String password = "UNSET";

    public Database (String pw) {
        password = pw;
        dbConnection = null;
        if (DBExists()) dbConnect(true); else dbCreate();
    }

    public static boolean DBDirExists() {
        File dir = new File(path);
        return dir.exists();
    }

    public static boolean DBExists() {
        File edb = new File(path + "/turtlenet.db.aes");
        File db = new File(path + "/turtlenet.db");
        return db.exists() || edb.exists();
    }

    public static boolean createDBDir() {
        return (new File(path)).mkdirs();
    }

    //Creates a database from scratch
    public void dbCreate() {
        Logger.write("INFO", "DB", "Creating database");
        try {
            if (!Database.DBDirExists())
                Database.createDBDir();
            dbConnect(false);
            for (int i = 0; i < DBStrings.createDB.length; i++)
                execute(DBStrings.createDB[i]);
        } catch (Exception e) {
            Logger.write("FATAL", "DB", "Failed to create databse: " + e);
        }
    }

    //Connects to a pre-defined database
    public boolean dbConnect(boolean dbexists) {
        if (dbexists)
            if (!Crypto.decryptDB(password))
                Logger.write("FATAL", "DB", "failed to decrypt database");

        Logger.write("INFO", "DB", "Connecting to database");
        try {
            Class.forName("org.sqlite.JDBC");
            dbConnection = DriverManager.getConnection("jdbc:sqlite:db/turtlenet.db");
            return true;
        } catch(Exception e) { //Exception logged to disk, program allowed to crash naturally
            Logger.write("FATAL", "DB", "Could not connect: " + e.getClass().getName() + ": " + e.getMessage() );
            return false;
        }
    }

    //Disconnects the pre-defined database
    public void dbDisconnect() {
        Logger.write("INFO", "DB", "Disconnecting from database");
        try {
            dbConnection.close();
        } catch(Exception e) { //Exception logged to disk, program allowed to continue
            Logger.write("FATAL", "DB", "Could not disconnect: " + e.getClass().getName() + ": " + e.getMessage() );
        }

        if (!Crypto.encryptDB(password))
            Logger.write("FATAL", "DB", "failed to encrypt database");
    }

    public void execute (String query) throws java.sql.SQLException {
        try {
            /*
            if (query.indexOf('(') != -1)
                Logger.write("VERBOSE", "DB", "execute(\"" + query.substring(0,query.indexOf('(')) + "...\")");
            else
                Logger.write("VERBOSE", "DB", "execute(\"" + query.substring(0,20) + "...\")");
            */
            Logger.write("VERBOSE", "DB", "execute(\"" + query + "\")");

            Statement statement = dbConnection.createStatement();
            statement.setQueryTimeout(30);
            dbConnection.setAutoCommit(false);
            statement.executeUpdate(query);
            dbConnection.commit();
            dbConnection.setAutoCommit(true);
        } catch (java.sql.SQLException e) {
            Logger.write("ERROR", "DB", "SQLException: " + e);
            throw e;
        }
    }

    public ResultSet query (String query) throws java.sql.SQLException {
        /*
        if (query.indexOf('(') != -1)
            Logger.write("VERBOSE", "DB", "query(\"" + query.substring(0,query.indexOf('(')) + "...\")");
        else
            Logger.write("VERBOSE", "DB", "query(\"" + query.substring(0,20) + "...\")");
        */
        Logger.write("VERBOSE", "DB", "query(\"" + query + "\")");

        try {
            Statement statement = dbConnection.createStatement();
            statement.setQueryTimeout(30);
            ResultSet r = statement.executeQuery(query);
            return r;
        } catch (java.sql.SQLException e) {
            Logger.write("RED", "DB", "Failed to query database: " + e);
            throw e;
        }
    }

    //Get from DB
    public String getPDATA(String field, PublicKey key) {
        Logger.write("VERBOSE", "DB", "getPDATA(" + field + ",...)");
        String value = "";
        try {
            String strKey = Crypto.encodeKey(key);
            String sqlStatement  = DBStrings.getPDATA.replace("__FIELD__", field);
            sqlStatement = sqlStatement.replace("__KEY__", strKey); //mods SQL template

            ResultSet results = query(sqlStatement);
            if(results.next())
                value = results.getString(field); //gets current value in 'field'
            else
                value = "<No Value>";
        } catch (java.sql.SQLException e) {
            Logger.write("ERROR", "DB", "SQLException: " + e);
        }

        if (value != null)
            return value;
        else
            return "<no value>";
    }

    //Set the CMD to POST in the Message constructor
    public Message[] getWallPost (PublicKey key) {
        Logger.write("VERBOSE", "DB", "getWallPost(...)");
        Vector<Message> posts = new Vector<Message>();
        try {
            String sqlStatement = DBStrings.getWallPostSigs.replace("__KEY__", Crypto.encodeKey(key) );
            ResultSet results = query(sqlStatement);

            while (results.next()) {
                Vector<String> visibleTo = new Vector<String>();
                ResultSet currentPost = query(DBStrings.getPost.replace("__SIG__", results.getString("sig")));
                ResultSet currentPostVisibleTo = query(DBStrings.getVisibleTo.replace("__SIG__", results.getString("sig")));
                while(currentPostVisibleTo.next())
                    visibleTo.add(currentPostVisibleTo.getString("key") );

                if(currentPost.next()) {
                    Message m = new MessageFactory().newPOST(currentPost.getString("msgText"), currentPost.getString
  ("recieverKey"), (visibleTo.toArray(new String[0])) );
                    m.timestamp = Long.parseLong(currentPost.getString("time"));
                    m.signature = currentPost.getString("sig");
                    m.command = "POST";
                    posts.add(m);
                }
            }
        } catch (java.sql.SQLException e) {
            Logger.write("ERROR", "DB", "SQLException: " + e);
        }

        return posts.toArray(new Message[0]);
    }

    public String getWallPostSender (String sig) {
        Logger.write("VERBOSE", "DB", "getWallPostSender(...)");
        try {
            ResultSet sendersKey = query(DBStrings.getPostSender.replace("__SIG__", sig));
            if (sendersKey.next())
                return sendersKey.getString("sendersKey");
            else
```

```java
                return "<POST DOESN'T EXIST>";
        } catch (java.sql.SQLException e) {
            Logger.write("ERROR", "DB", "SQLException: " + e);
            return "ERROR";
        }
    }

    public Message[] getComments (String sig) {
        Vector<Message> comments = new Vector<Message>();
        Logger.write("VERBOSE", "DB", "getComments(...)");

        try {
            ResultSet commentSet = query(DBStrings.getComments.replace("__PARENT__", sig));
            while (commentSet.next()) {
                Message cmnt = new MessageFactory().newCMNT(sig, commentSet.getString("msgText"));
                cmnt.timestamp = Long.parseLong(commentSet.getString("creationTime"));
                cmnt.signature = commentSet.getString("sig");
                comments.add(cmnt);
            }
        } catch (java.sql.SQLException e) {
            Logger.write("ERROR", "DB", "SQLException: " + e);
        }

        return comments.toArray(new Message[0]);
    }

    public Long timeMostRecentWallPost (PublicKey key) {
        Logger.write("VERBOSE", "DB", "timeMostRecentWallPost(...)");
        try {
            ResultSet mostRecent = query(DBStrings.mostRecentWallPost.replace("__KEY__", Crypto.encodeKey(key)));
            if (mostRecent.next())
                return Long.parseLong(mostRecent.getString("maxtime"));
        } catch (java.sql.SQLException e) {
            Logger.write("ERROR", "DB", "SQLException: " + e);
        }
        return 0L;
    }

    public boolean isLiked (String sig) {
        Logger.write("VERBOSE", "DB", "isLiked(...)");
        int ret = 0;

        try {
            ResultSet row = query(DBStrings.getLike.replace("__SIG__", sig));
            return row.next();
        } catch (java.sql.SQLException e) {
            Logger.write("ERROR", "DB", "SQLException: " + e);
        }

        return false;
    }

    //Return all conversations
    public Conversation[] getConversations () {
        Vector<Conversation> convoList = new Vector<Conversation>();
        Logger.write("VERBOSE", "DB", "getConversations()");

        try {
            ResultSet convoSet = query(DBStrings.getConversations);
            while (convoSet.next())
                convoList.add(getConversation(convoSet.getString("convoID")));
        } catch (java.sql.SQLException e) {
            Logger.write("ERROR", "DB", "SQLException: " + e);
        }

        return convoList.toArray(new Conversation[0]);
    }

    //Get keys of all people in the given conversation
    public PublicKey[] getPeopleInConvo (String sig) {
        Logger.write("VERBOSE", "DB", "getPeopleInConvo(...)");
        Vector<PublicKey> keys = new Vector<PublicKey>();

        try {
            ResultSet keySet = query(DBStrings.getConversationMembers.replace("__SIG__", sig));
            while (keySet.next())
                keys.add(Crypto.decodeKey(keySet.getString("key")));
        } catch (java.sql.SQLException e) {
            Logger.write("ERROR", "DB", "SQLException: " + e);
        }

        return keys.toArray(new PublicKey[0]);
    }

    //Reurn a conversation object
    public Conversation getConversation (String sig) {
        Logger.write("VERBOSE", "DB", "getConversation(...)");
        try {
            ResultSet convoSet = query(DBStrings.getConversation.replace("__SIG__", sig));
            if(convoSet.next()) {
                String timestamp = convoSet.getString("time");
                ResultSet messages = query(DBStrings.getConversationMessages.replace("__SIG__", sig));
                String firstMsg;
                if (messages.next())
                    firstMsg = messages.getString("msgText");
                else
                    firstMsg = "<no messages yet>";
                PublicKey[] keys = getPeopleInConvo(sig);
                String[] keystrings = new String[keys.length];
                String[] users = new String[keys.length];
                for (int i = 0; i < keys.length; i++) {
                    keystrings[i] = Crypto.encodeKey(keys[i]);
                    users[i] = getName(keys[i]);
                }
                return new Conversation(sig, timestamp, firstMsg, users, keystrings);
            } else {
                Logger.write("WARNING", "DB", "getConversation(...) empty conversation: " + sig);
            }
        } catch (java.sql.SQLException e) {
            Logger.write("ERROR", "DB", "SQLException: " + e);
        }
        return new Conversation();
    }

    //Return all messages in a conversation
    //{{username, time, msg}, {username, time, msg}, etc.}
    //Please order it so that element 0 is the oldest message
    public String[][] getConversationMessages (String sig) {
        Logger.write("VERBOSE", "DB", "getConversationMessages(...)");
        Vector<String[]> messagesList = new Vector<String[]>();

        try {
            ResultSet messageSet = query(DBStrings.getConversationMessages.replace("__SIG__", sig));
            while(messageSet.next() ) {
                String[] message = new String[3];
                message[0] = getName(Crypto.decodeKey(messageSet.getString("sendersKey")));
                message[1] = messageSet.getString("time");
                message[2] = messageSet.getString("msgText");

                messagesList.add(message);
            }
        } catch (java.sql.SQLException e) {
            Logger.write("ERROR", "DB", "SQLException: " + e);
        }

        return messagesList.toArray(new String[0][0]);
    }

    //If multiple people have the same username then:
    //Logger.write("FATAL", "DB", "Duplicate usernames");
    //System.exit(1);
    public PublicKey getKey (String userName) {
        Logger.write("VERBOSE", "DB", "getKey(" + userName + ")");
        int nameCount = 0;
        String key = "<No Key>";

        try {
            ResultSet results = query(DBStrings.getKey.replace("__USERNAME__", userName) );
            while(results.next()) {
                nameCount++;
                key = results.getString("key");
            }
        } catch (java.sql.SQLException e) {
            Logger.write("ERROR", "DB", "SQLException: " + e);
        }

        if(nameCount == 0)
            Logger.write("ERROR", "DB", "getKey(" +  userName + ") - No keys found for userName");
        else if (nameCount > 1 )
            Logger.write("ERROR", "DB", "getKey(" + userName + ") - Multple userNames found for key; Server OPs are evil!");

        return Crypto.decodeKey(key);
    }

    public boolean canSeePDATA (String category) {
        Logger.write("VERBOSE", "DB", "canSeePDATA()");

        try {
            ResultSet categorySet = query(DBStrings.canSeePDATA.replace("__CATID__", category));
            if (categorySet.next()) {
                return categorySet.getInt("canSeePDATA") == 1 ? true : false;
            }
        } catch (java.sql.SQLException e) {
            Logger.write("ERROR", "DB", "SQLException: " + e);
        }

        return false;
    }

    //Return the name of each member and if it can see your profile info
    //In this format: {{"friends", "false"}, {"family", "true"}, etc.}
    public String[][] getCategories () {
        Logger.write("VERBOSE", "DB", "getCategories()");
        Vector<String[]> catList = new Vector<String[]>();
        String catName;
        String canSeePDATA;
```

```java

        try {
            ResultSet categorySet = query(DBStrings.getCategories);
            while(categorySet.next() ) {
                String[] category = new String[2];
                category[0] = categorySet.getString("catID");
                category[1] = categorySet.getInt("canSeePDATA") == 1 ? "true" : "false";
                catList.add(category);
            }
        } catch (java.sql.SQLException e) {
            Logger.write("ERROR", "DB", "SQLException: " + e);
        }

        Logger.write("VERBOSE", "DB", "getCategories() returning " + catList.toArray().length + " categories");
        return catList.toArray(new String[0][0]);
    }

    //Return the keys of each member of the category
    //if(category.equals("all")) //remember NEVER to compare strings with ==
    //    return every key you know about
    public PublicKey[] getCategoryMembers (String catID) {
        Logger.write("VERBOSE", "DB", "getCategoryMembers(" + catID + ")");
        String queryStr = "";

        if(catID.toLowerCase().equals("all"))
            queryStr = DBStrings.getAllKeys;
        else
            queryStr = DBStrings.getMemberKeys.replace("__CATNAME__", catID);

        Vector<PublicKey> keyList = new Vector<PublicKey>();

        try {
            ResultSet keySet = query(queryStr);
            while(keySet.next()) {
                if(catID.toLowerCase().equals("all"))
                    keyList.add(Crypto.decodeKey(keySet.getString("key")));
                else
                    keyList.add(Crypto.decodeKey(keySet.getString("userKey")));
            }
        } catch (java.sql.SQLException e) {
            Logger.write("ERROR", "DB", "SQLException: " + e);
        }

        Logger.write("VERBOSE", "DB", "getCategoryMembers(" + catID + ") returning " + keyList.toArray().length + " members");
        return keyList.toArray(new PublicKey[0]);
    }

    //Given the sig of a post or comment return the keys which can see it
    public PublicKey[] getVisibilityOfParent(String sig) {
        Logger.write("VERBOSE", "DB", "getVisibilityOfParent(" + sig + ")");

        try {
            ResultSet postWithSig = query(DBStrings.getPost.replace("__SIG__", sig));
            if (postWithSig.next()) { //sig is a post
                Logger.write("VERBOSE", "DB", "parent is a wall post: " + sig);
                return getPostVisibleTo(sig);
            } else { //sig is a comment
                ResultSet commentWithSig = query(DBStrings.getComment.replace("__SIG__", sig));
                if (commentWithSig.next())
                    return getVisibilityOfParent(commentWithSig.getString("parent"));
                else
                    Logger.write("ERROR", "DB", "getVisibilityOfParent has no root");
            }
        } catch (java.sql.SQLException e) {
            Logger.write("ERROR", "DB", "SQLException: " + e);
        }

        return null;
    }

    public PublicKey[] getPostVisibleTo (String sig) {
        Logger.write("VERBOSE", "DB", "getVisibleTo(...)");
        Vector<PublicKey> keyList = new Vector<PublicKey>();

        try {
            ResultSet keyRows = query(DBStrings.getVisibleTo.replace("__SIG__", sig));
            while(keyRows.next())
                keyList.add(Crypto.decodeKey(keyRows.getString("key")));
        } catch (java.sql.SQLException e) {
            Logger.write("ERROR", "DB", "SQLException: " + e);
        }

        return keyList.toArray(new PublicKey[0]);
    }

    //In the case of no username for the key: "return Crypto.encode(k);"
    public String getName (PublicKey key) {
        Logger.write("VERBOSE", "DB", "getName(...)");
        String name = "";

        try {
            ResultSet nameRow = query(DBStrings.getName.replace("__KEY__", Crypto.encodeKey(key)));
            if (nameRow.next())
```

```java
                name = nameRow.getString("username");
        } catch (java.sql.SQLException e) {
            Logger.write("ERROR", "DB", "SQLException: " + e);
        }

        if (name != null)
            return name;
        else
            return "<no username>";
    }

    //"What key signed this message"
    public PublicKey getSignatory (Message m) {
        Logger.write("VERBOSE", "DB", "getSignatory(...)");
        try {
            ResultSet keys = query(DBStrings.getAllKeys);
            while (keys.next())
                if (Crypto.verifySig(m, Crypto.decodeKey(keys.getString("key"))))
                    return Crypto.decodeKey(keys.getString("key"));
        } catch (java.sql.SQLException e) {
            Logger.write("ERROR", "DB", "SQLException: " + e);
        }
        Logger.write("WARNING", "DB", "getSignatory() could not find signatory");
        return null;
    }

    //Add to DB
    public boolean addPost (Message post) {
        Logger.write("VERBOSE", "DB", "addPost(...)");

        try {
            execute(DBStrings.addPost.replace("__SIG__", post.getSig())
                                     .replace("__msgText__", post.POSTgetText())
                                     .replace("__time__", Long.toString(post.getTimestamp()))
                                     .replace("__recieverKey__", post.POSTgetWall())
                                     .replace("__sendersKey__", Crypto.encodeKey(getSignatory(post))));
            String[] visibleTo = post.POSTgetVisibleTo();
            for (int i = 0; i < visibleTo.length; i++)
                execute(DBStrings.addPostVisibility.replace("__postSig__", post.getSig()).replace("__key__", visibleTo[i]));
            return true;
        } catch (java.sql.SQLException e) {
            Logger.write("ERROR", "DB", "SQLException: " + e);
            return false;
        }
    }

    public boolean addKey (Message msg) {
        return addKey(Crypto.decodeKey(msg.ADDKEYgetKey()));
    }

    public boolean addKey (PublicKey k) {
        Logger.write("VERBOSE", "DB", "addKey(...)");

        try {
            execute(DBStrings.addKey.replace("__key__", Crypto.encodeKey(k)));
            boolean ret = validateClaims(k);
            if (!calcRevocationKeys(k))
                ret = false;
            return ret;
        } catch (java.sql.SQLException e) {
            Logger.write("ERROR", "DB", "SQLException: " + e);
        }

        return false;
    }

    //Update k's username by validating claims
    public boolean validateClaims(PublicKey k) {
        if (k == null) {
            Logger.write("ERROR", "DB", "validateClaims(...) called with null key");
            return false;
        }

        Logger.write("VERBOSE", "DB", "validateClaims(...)");

        try {
            ResultSet claimSet = query(DBStrings.getClaims);
            while (claimSet.next()) {
                Message msg = new Message("CLAIM",
                                          claimSet.getString("name"),
                                          Long.parseLong(claimSet.getString("claimTime")),
                                          claimSet.getString("sig"));

                Logger.write("VERBOSE", "DB", "Considering Claim for name: \"" + claimSet.getString("name") + "\"");
                Logger.write("VERBOSE", "DB", "                       time: \"" + Long.toString(Long.parseLong
(claimSet.getString("claimTime"))) + "\"");
                Logger.write("VERBOSE", "DB", "                        sig: \"" + claimSet.getString("sig") + "\"");

                PublicKey signatory = getSignatory(msg);
                if (signatory != null && signatory.equals(k)) {
                    execute(DBStrings.newUsername.replace("__name__", msg.CLAIMgetName()).replace("__key__", Crypto.encodeKey
(k)));
                    execute(DBStrings.removeClaim.replace("__sig__", msg.getSig()));
                    Logger.write("INFO", "DB", "Claim for " + msg.CLAIMgetName() + " verified");
                }
            }
        } catch (java.sql.SQLException e) {
            Logger.write("ERROR", "DB", "SQLException: " + e);
            return false;
        }
        return true;
    }

    //update keys column in revocations
    public boolean calcRevocationKeys (PublicKey k) {
        if (k == null) {
            Logger.write("ERROR", "DB", "calcRevocationKeys(...) called with null key");
            return false;
        }

        Logger.write("VERBOSE", "DB", "calcRevocationKeys(...)");

        try {
            ResultSet revocationSet = query(DBStrings.getRevocations);
            while (revocationSet.next()) {
                Message msg = new Message("REVOKE",
                                          revocationSet.getString("timeOfLeak"),
                                          Long.parseLong(revocationSet.getString("creationTime")),
                                          revocationSet.getString("sig"));
                PublicKey signer = getSignatory(msg);
                if (signer != null && signer.equals(k)) {
                    execute(DBStrings.updateRevocationKey.replace("__KEY__", Crypto.encodeKey(k))
                                                         .replace("__SIG__", revocationSet.getString("sig")));
                }
            }
        } catch (java.sql.SQLException e) {
            Logger.write("ERROR", "DB", "SQLException: " + e);
            return false;
        }
        return true;
    }

    //if this key has already claimed a name, forget the old one
    public boolean addClaim (Message claim) {
        Logger.write("VERBOSE", "DB", "addClaim("+ claim.CLAIMgetName() +")");

        try {
            execute(DBStrings.addClaim.replace("__sig__", claim.getSig())
                                      .replace("__name__", claim.CLAIMgetName())
                                      .replace("__time__", Long.toString(claim.getTimestamp())));

            ResultSet everyone = query(DBStrings.getAllKeys);
            while (everyone.next())
                    validateClaims(Crypto.decodeKey(everyone.getString("key")));
        } catch (java.sql.SQLException e) {
            Logger.write("ERROR", "DB", "SQLException: " + e);
            return false;
        }
        return true;
    }

    public boolean addRevocation (Message revocation) {
        Logger.write("VERBOSE", "DB", "-------addRevocation(...)-------");

        try {
            execute(DBStrings.addRevocation.replace("__key__", Crypto.encodeKey(getSignatory(revocation)))
                                           .replace("__sig__", revocation.getSig())
                                           .replace("__time__", Long.toString(revocation.REVOKEgetTime()))
                                           .replace("__creationTime__", Long.toString(revocation.getTimestamp())));
            return eraseContentFrom(getSignatory(revocation));
        } catch (java.sql.SQLException e) {
            Logger.write("ERROR", "DB", "SQLException: " + e);
            return false;
        }
    }

    public boolean isRevoked (PublicKey key) {
        Logger.write("VERBOSE", "DB", "isRevoked(...)");

        try {
            return query(DBStrings.isRevoked.replace("__KEY__", Crypto.encodeKey(key))).next();
        } catch (java.sql.SQLException e) {
            Logger.write("ERROR", "DB", "SQLException: " + e);
            return false;
        }
    }

    public boolean eraseContentFrom(PublicKey key) {
        Logger.write("VERBOSE", "DB", "-------eraseContentFrom(...)-------");
        String keyStr = Crypto.encodeKey(key);

        try {
            execute(DBStrings.removeMessageAccess.replace("__KEY__", keyStr));
            execute(DBStrings.removeMessages.replace("__KEY__", keyStr));
            execute(DBStrings.removePosts.replace("__KEY__", keyStr));
            execute(DBStrings.removePostVisibility.replace("__KEY__", keyStr));
            execute(DBStrings.removeUser.replace("__KEY__", keyStr));
            execute(DBStrings.removeFromCategories.replace("__KEY__", keyStr));
            execute(DBStrings.removeLikes.replace("__KEY__", keyStr));
            execute(DBStrings.removeComments.replace("__KEY__", keyStr));
            execute(DBStrings.removeEvents.replace("__KEY__", keyStr));
        } catch (java.sql.SQLException e) {
            Logger.write("ERROR", "DB", "SQLException: " + e);
            return false;
        }

        return true;
    }

    public boolean addPDATA (Message update) {
        Logger.write("VERBOSE", "DB", "addPDATA(...)");
        boolean ret = true;

        String[][] updates = update.PDATAgetValues();
        for (int i = 0; i < updates.length; i++)
            if (!updatePDATA(updates[i][0], updates[i][1], getSignatory(update)))
                ret = false;

        return ret;
    }

    public boolean updatePDATA (String field, String value, PublicKey k) {
        Logger.write("VERBOSE", "DB", "updatePDATA(" + field + ", " + value + ", ...)");

        try {
            execute(DBStrings.addPDATA.replace("__field__", field)
                                      .replace("__value__", value)
                                      .replace("__key__", Crypto.encodeKey(k)));
        } catch (java.sql.SQLException e) {
            Logger.write("ERROR", "DB", "SQLException: " + e);
            return false;
        }

        return true;
    }

    public boolean addConvo (Message convo) {
        Logger.write("VERBOSE", "DB", "addConvo(...)");

        try {
            execute(DBStrings.addConvo.replace("__sig__", convo.getSig())
                                      .replace("__time__", Long.toString(convo.getTimestamp())));
            String[] keys = convo.CHATgetKeys();
            for (int i = 0; i < keys.length; i++) {
                execute(DBStrings.addConvoParticipant.replace("__sig__", convo.getSig())
                                                     .replace("__key__", keys[i]));
            }
        } catch (java.sql.SQLException e) {
            Logger.write("ERROR", "DB", "SQLException: " + e);
            return false;
        }

        return true;
    }

    public boolean addMessageToChat (Message msg) {
        Logger.write("VERBOSE", "DB", "addMessageToChat(...)");

        try {
            boolean duplicate = false;

            String[][] messagesInConvo = getConversationMessages(msg.PCHATgetConversationID());
            for (int i = 0; i < messagesInConvo.length; i++)
                if (messagesInConvo[i][1].equals(Long.toString(msg.getTimestamp())) && messagesInConvo[i][2].equals
(msg.PCHATgetText()))
                    duplicate = true;

            if (!duplicate) {
                execute(DBStrings.addMessageToConvo.replace("__convoID__", msg.PCHATgetConversationID())
                                                   .replace("__sendersKey__", Crypto.encodeKey(getSignatory(msg)))
                                                   .replace("__msgText__", msg.PCHATgetText())
                                                   .replace("__time__", Long.toString(msg.getTimestamp())));
            }
        } catch (java.sql.SQLException e) {
            Logger.write("ERROR", "DB", "SQLException: " + e);
            return false;
        }

        return true;
    }

    public boolean addComment (Message comment) {
        Logger.write("VERBOSE", "DB", "addComment(...)");

        try {
            execute(DBStrings.addComment.replace("__sig__", comment.getSig())
                                        .replace("__msgText__", comment.CMNTgetText())
                                        .replace("__parent__", comment.CMNTgetItemID())
                                        .replace("__commenterKey__", Crypto.encodeKey(getSignatory(comment)))
                                        .replace("__senderKey__", Crypto.encodeKey(getSignatory(comment)))
                                        .replace("__creationTime__", Long.toString(comment.getTimestamp())));
        } catch (java.sql.SQLException e) {
            Logger.write("ERROR", "DB", "SQLException: " + e);
            return false;
        }

        return true;
    }

    public boolean addLike (Message like) {
        Logger.write("VERBOSE", "DB", "addLike(...)");

        try {
            execute(DBStrings.addLike.replace("__likerKey__", Crypto.encodeKey(getSignatory(like)))
                                     .replace("__parent__", like.LIKEgetItemID()));
        } catch (java.sql.SQLException e) {
            Logger.write("ERROR", "DB", "SQLException: " + e);
            return false;
        }

        return true;
    }

    public boolean addEvent (Message event) {
        Logger.write("VERBOSE", "DB", "addEvent(...)");
        try {
            execute(DBStrings.addEvent.replace("__sig__", event.getSig())
                                      .replace("__startTime__", Long.toString(event.EVNTgetStart()))
                                      .replace("__endTime", Long.toString(event.EVNTgetEnd()))
                                      .replace("__creatorKey__", Crypto.encodeKey(getSignatory(event)))
                                      .replace("__accepted__", "0")
                                      .replace("__name__", event.EVNTgetName())
                                      .replace("__creationTime__", Long.toString(event.getTimestamp())));
        } catch (java.sql.SQLException e) {
            Logger.write("ERROR", "DB", "SQLException: " + e);
            return false;
        }

        return true;
    }

    public boolean acceptEvent (String sig) {
        Logger.write("VERBOSE", "DB", "acceptEvent(...)");
        try {
            execute(DBStrings.acceptEvent.replace("__sig__", sig));
        } catch (java.sql.SQLException e) {
            Logger.write("ERROR", "DB", "SQLException: " + e);
            return false;
        }

        return true;
    }

    public boolean declineEvent (String sig) {
        Logger.write("VERBOSE", "DB", "declineEvent(...)");
        try {
            execute(DBStrings.declineEvent.replace("__sig__", sig));
        } catch (java.sql.SQLException e) {
            Logger.write("ERROR", "DB", "SQLException: " + e);
            return false;
        }

        return true;
    }

    public boolean updatePDATApermission (Message msg) {
        return updatePDATApermission(msg.UPDATECATgetName(), msg.UPDATECATgetValue());
    }

    public boolean updatePDATApermission (String category, boolean value) {
        Logger.write("VERBOSE", "DB", "updatePDATApermission(...)");
        try {
            execute(DBStrings.updatePDATApermission.replace("__catID__", category)
                                                   .replace("__bool__", value?"1":"0"));
        } catch (java.sql.SQLException e) {
            Logger.write("ERROR", "DB", "SQLException: " + e);
            return false;
        }

        return true;
    }

    public PublicKey[] keysCanSeePDATA () {
        Logger.write("VERBOSE", "DB", "keysCanSeePDATA()");
        Vector<PublicKey> keys = new Vector<PublicKey>();

        try {
            ResultSet categories = query(DBStrings.categoriesCanSeePDATA);
            while (categories.next()) {
                String catname = categories.getString("catID");
                PublicKey[] memberKeys = getCategoryMembers(catname);
                for (int i = 0; i < memberKeys.length; i++)
```

```java
                    if (!keys.contains(memberKeys[i]))
                        keys.add(memberKeys[i]);
            }
        } catch (java.sql.SQLException e) {
            Logger.write("ERROR", "DB", "SQLException: " + e);
        }

        return keys.toArray(new PublicKey[0]);
    }

    //no duplicate names
    public boolean addCategory (Message msg) {
        return addCategory(msg.ADDCATgetName(), msg.ADDCATgetValue());
    }

    public boolean addCategory (String name, boolean can_see_private_details) {
        Logger.write("VERBOSE", "DB", "addCategory(...)");
        try {
            execute(DBStrings.addCategory.replace("__catID__", name)
                                         .replace("__canSeePDATA__", can_see_private_details?"1":"0"));
        } catch (java.sql.SQLException e) {
            Logger.write("ERROR", "DB", "SQLException: " + e);
            return false;
        }

        return true;
    }

    public boolean addToCategory (Message msg) {
        return addToCategory(msg.ADDTOCATgetName(), Crypto.decodeKey(msg.ADDTOCATgetKey()));
    }

    public boolean addToCategory (String category, PublicKey key) {
        Logger.write("VERBOSE", "DB", "addToCategory(" + category + ", ...)");

        PublicKey[] members = getCategoryMembers(category);
        if (Arrays.asList(members).contains(key)) {
            return false;
        }

        try {
            execute(DBStrings.addToCategory.replace("__catID__", category)
                                           .replace("__key__", Crypto.encodeKey(key)));
        } catch (java.sql.SQLException e) {
            Logger.write("ERROR", "DB", "SQLException: " + e);
            return false;
        }

        return true;
    }

    public boolean removeFromCategory (Message msg) {
        return removeFromCategory(msg.REMFROMCATgetCategory(), Crypto.decodeKey(msg.REMFROMCATgetKey()));
    }

    public boolean removeFromCategory (String category, PublicKey key) {
        Logger.write("VERBOSE", "DB", "removeFromCategory(" + category + ", ...)");
        try {
            execute(DBStrings.removeFromCategory.replace("__catID__", category)
                                                .replace("__key__", Crypto.encodeKey(key)));
        } catch (java.sql.SQLException e) {
            Logger.write("ERROR", "DB", "SQLException: " + e);
            return false;
        }

        return true;
    }

    public boolean like (String sig) {
        Logger.write("VERBOSE", "DB", "like(...)");
        try {
            execute(DBStrings.addLike.replace("__parent__", sig)
                                     .replace("__likerKey__", Crypto.encodeKey(Crypto.getPublicKey())));
        } catch (java.sql.SQLException e) {
            Logger.write("ERROR", "DB", "SQLException: " + e);
            return false;
        }

        return true;
    }

    public boolean unlike (String sig) {
        Logger.write("VERBOSE", "DB", "like(...)");
        try {
            execute(DBStrings.removeLike.replace("__parent__", sig)
                                     .replace("__likerKey__", Crypto.encodeKey(Crypto.getPublicKey())));
        } catch (java.sql.SQLException e) {
            Logger.write("ERROR", "DB", "SQLException: " + e);
            return false;
        }

        return true;
    }
```

```java
}
```