Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

CSU34031 Advanced Telecommunications Assignment #2: Secure Social

Jack Gilbride, 17340868

April 14, 2020

Contents

1 Introduction

The task of this assignment was to develop a secure social media application. In the secure social application, only members of a secure group should be able to decrypt each other's posts. To all other users of the system the post should appear as ciphertext. The assignment required the design and implementation of a suitable key management system that allows any member of the group to share social media messages securely, and allows the addition and removal of people from a group. We were free to implement the application for a desktop or mobile platform and make use of any open source cryptographic libraries.
In this report I will document my approach towards this assignment, starting with the theory of the topic, then describing my own implementation in Java. I will finish with a summary. This will be followed by an appendix of the code deliverables produced for the assignment.

2 Theory of Topic

In this section I will describe the concepts that were used to realise a solution to my final web proxy server. In particular I will discuss encryption concepts, including Symmetric-Key Cryptography and Asymmetric-Key Cryptography.

2.1 Symmetric-Key Cryptography

Symmetric-Key Encryption is a type of cryptography based on the sender and the receiver of a message knowing and using a shared secret key. The sender uses the secret key to encrypt the message and the receiver uses the same secret key to decrypt the message. The main problem with this is getting the sender and receiver to agree on a secret key without anyone else finding out, especially if there is no cryptosystem in place to begin with.

2.2 Advanced Encryption Standard

Advanced Encryption Standard (AES) uses a symmetric cipher with variable key and block sizes of 128, 192 and 256 bits. The cipher consists of between 10 or 14 rounds depending on the key length & block length. A plaintext block X undergoes n rounds of operations to produce an output block Y. Each operation is based on the value of the n^{th} round key. Round keys are derived from the Cipher Key by first expanding the key, then selecting parts of the expanded key for each round. AES has the following advantages:

- It is secure. No backdoors are known for AES. This means that without the key, ciphertext cannot be decoded.
- There is support for fast AES encryption and decryption in software.
- The large key and block sizes make it computationally infeasible to break by brute force. A device that could check 10^{18} AES keys per second would require about 3×10^{51} years to exhaust the 256-bit key space.

2.3 Asymmetric-Key Cryptography

Asymmetric-key cryptography solves the key management problem associated with symmetric key cryptosystems. It is otherwise known as public key cryptography because each person generates a pair of keys (a public and private key). The public key is published and widely distributed, while the private key is kept secret. In this system, if Alice wants to send Bob an encrypted message, she can encrypt it using Bob's public key. Bob can then decrypt the message using his own private key. This solves the key management problem as Bob never needs to provide his private key, but can still be sent encrypted messages. Asymmetric-key cryptosystems have the following properties:

- It must be computationally easy to encipher or decipher a message given the appropriate key.
- It must be computationally infeasible to derive the private key from the public key.

2.4 Electronic Code Book vs Cipher Block Chaining

Two possible modes of operation for block ciphers are Electronic Code Book (ECB) and Cipher Block Chaining (CBC). In EBC each block is encrypted independently of all other blocks. This means that identical plain text blocks result in identical ciphertext blocks. There is a fundamental flaw in this; any pattern in the original plain text would be replicated in the cipher text. Using this mode of operation could leave the encryption open to an attack which analyses patterns in the cipher text to decrypt individual cipher text blocks. These individual decodings would also give the attacker a better chance at decrypting the rest of the cipher. In CBC, in contrast, each block of plaintext is XORed with previous ciphertext block before being encrypted. If we encrypt a string of blocks once with a first initialization vector (IV) and a second time with a different IV, the two resulting ciphertext sequences will look completeley unreleasted to each other. The initialization vector does not have to be kept secret for this to work.

2.5 RSA

RSA is the de-facto standard algorithm for implementing asymmetric-key cryptography. Its security is based on the difficulty of factoring very large numbers. RSA uses a key size of either 1024, 2048, 3072 or 7680 bits. The algorithm works as follows:

- Choose two large distinct primes p and q.
- Compute $\phi(n) = (p-1) \cdot (q-1)$.
- Randomly choose an encryption key e, less than n that has no common factors with $\phi(n)$. e and $\phi(n)$ are relatively prime.
- Compute the decryption key d such that $d \equiv e^{-1} mod \phi(n)$.

The numbers e and n are the public key and the number d is the private key. For decryption and encryption, the plaintext message is broken into a number of blocks, and each block is represented as an integer. To encrypt we use the following formula $CiphertextBlock = (PlaintextBlock)^e mod(n)$. To decrypt we use the formula $PlaintextBlock = (CiphertextBlock)^d mod(n)$.

3 Implementation

Having investigated the theory of the topic, I decided on my approach to the problem:

- Each group has a symmetric key. This symmetric key is used to encode all messages posted within that group.
- Each user has a public key and a private key. The public key is available for everyone to see and the private key is only seen by that user.
- An admin is a special user who can decide whether to add or remove a user to/from a group.
- When a user is put into a group, the group encodes its symmetric key with the user's public key using RSA. The group then sends this encoding to the user.
- The user receives this encoding, and decodes it with their own private key using RSA. The decoded message is the symmetric key for the group.
- The user can use this symmetric key to decode it's group member's messages using AES.
- When a user is removed from a group, the symmetric key for that group changes. It is then sent out to each group member as discussed previously.

Unfortunately due to time constraints I did not get to fully realize my solution. However, this model of the problem was a vital part of the assignment as it made me realize how I could utilize the concepts learnt in lectures to solve the problem.

3.1 Interacting with the Database

My initial efforts in this assignment were spent setting up a database client, which would interact with MongoDB to read and write all the relevant data for the assignment. JSON schema were made to represent key concepts that we would want to store in the database, such as users and groups. I managed to successfully set up the database to allow for a scalable application, and interact with it. This can be seen in *DBClient.java* in the appendix. Unfortunately the assignment did not get to a stage where this functionality could be tied into the final application. The remainder of this report will focus on the code written to realise some of the concepts described in the theory of the topic.

3.2 AES

As mentioned already, an important aspect of my solution would be encryption and decryption in AES. This would be important functionality to decode a group member's message when the key is known. It was therefore necessary for a user to have two functions; *encryptAES* and *decryptAES*. To do this I used the *javax.crypto* library. As it is a commonly used library, I knew it would be tried and tested so would be secure. The *encrypt* function takes a message in the form of a String, and a secret key. It firstly converts the message to an array of bytes so it can process it for encryption. The cipher chosen is AES for the benefits discussed previously. The key generated comes from the group, and has the length 256, to make breaking

this cipher by brute force computationally infeasible. CBC is used as the mode of operation, also due to benefits discussed already. For this reason we must choose a unique initialization vector (IV). The cipher chooses its own IV and, given the key, is able to encrypt the plaintext passed to the function. This IV will need to be known to decrypt the cipher later, so we append it to the front of the payload. So the string returned by the payload is the encryption of the original message, prefixed with the initialization vector needed to decrypt it. Note that we can simply append IV to the front of the encryption as it does not need to be kept secret; it only needs to be unique (a property which we get by allowing the cipher to generate it).

```
/*
 * Encrypt a message with AES using a secret key. Returns a string which is
    the
 * initialization vector appended to the encoded text.
 */
protected String encryptAES(String message, SecretKey key) {
        try {
                // Convert the message to bytes for us to work with
                byte[] plainText = message.getBytes();
                // Initialize the cipher using the secret key
                SecretKeySpec keySpec = new SecretKeySpec(key.getEncoded(),
                    "AES");
                Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
                cipher.init(Cipher.ENCRYPT_MODE, keySpec);
                // Encrypt the message
                byte[] encrypted = cipher.doFinal(plainText);
                // Append the initialisation vector to the start of the
                    payload
                byte[] iv = cipher.getIV();
                byte[] payload = new byte[iv.length + encrypted.length];
                System.arraycopy(iv, 0, payload, 0, iv.length);
                System.arraycopy(encrypted, 0, payload, iv.length,
                    encrypted.length);
                // Return the payload
                String cipherText = new String(payload);
                return cipherText;
        } catch (Exception e) {
                e.printStackTrace();
                return null;
        }

}
```

Listing 1: AES Encryption.

As AES is a symmetric encryption method, the function for decryption is essentially the inverse for that of encryption. Firstly, we must convert the string to a format that the decryption can work with; a byte array. The initalization is taken from the front of the array and the remaining bytes are the encrypted message. We again use AES with CBC. As this is symmetric, the key used to decrypt the message will be the same as the key used to encrypt it. The cipher is decoded using the symmetric key and initialization vector. The cipher has now been decoded and we may return it, converting it to its desired form (String).

```
protected String decryptAES(String message, SecretKey key) {
        try {
                // Convert the message to bytes for us to work with
                byte[] payload = message.getBytes();
                // Take off the initialization vector which was used to
                    encrypt
        byte[] iv = new byte[16];
```

```
                byte[] cipherText = new byte[payload.length - iv.length];
        System.arraycopy(payload, 0, iv, 0, 16);
        System.arraycopy(payload, iv.length, cipherText, 0,
            cipherText.length);
                // Decrypt the message now that we know the initialization
                    vector
        SecretKeySpec keySpec = new SecretKeySpec(key.getEncoded(), "AES");
                Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
                cipher.init(Cipher.DECRYPT_MODE, keySpec, new
                    IvParameterSpec(iv));
                // Convert to the string form that we want and return
                String decryptedText = new String(cipher.doFinal(cipherText));
                return decryptedText;
        } catch (Exception e) {
                e.printStackTrace();
                return null;
        }
}
```

Listing 2: Decrypting AES

3.3 RSA

While we could use the advantages of symmetric-key encryption among a group, the main problem that we encounter is how to get this symmetric key to members of the group, without people from outside of the group knowing. For this we can use an asymmetric-key encryption method. Due to it being the de-facto standard, RSA was used as the asymmetric key encryption method in this assignment. We could use RSA to add a user to a group. A user's membership of a group would depend on whether they know the group's symmetric key or not. The group's symmetric key could be encoded by the group with the user's public key, and sent to the user. That way, only the intended user could decode the message with their own private key, thus finding out the group's symmetric key. In order for a user to use RSA, they must be able to generate a key pair. This is done using the *java.security* library. Again this is done using a trusted library as it has become a tried and tested method. We use a KeyPairGenerator and save the keys in the user object. We specifiy a key length of 2048 as it is a recommended key length in RSA.

```
KeyPairGenerator keyPairGenerator;
keyPairGenerator = KeyPairGenerator.getInstance("RSA");
keyPairGenerator.initialize(2048);
KeyPair kp = keyPairGenerator.genKeyPair();
this.publicKey = kp.getPublic();
this.privateKey = kp.getPrivate();
```

Listing 3: Generating a key pair.

Now that the user has the ability to generate keys, it is important for a group to be able to encrypt a message with the user's public key. Again, we use functionality which is built into Java. Assuming that the group has a textual representation of the public key, it must first be converted into a public key object that the cipher can work with. The function generates this object from the string. We then use the same Cipher class that we worked with previously, to encrypt the message. Finally we return it in the form of a String as before.

```
protected String encryptRSA(String message, String publicKey) {
        try {
                X509EncodedKeySpec keySpec = new
                    X509EncodedKeySpec(Base64.getDecoder().decode(publicKey.getBytes()
                PublicKey keyObject =
                    KeyFactory.getInstance("RSA").generatePublic(keySpec);
```

```
                Cipher cipher = Cipher.getInstance("RSA/ECB/PKCS1Padding");
                cipher.init(Cipher.ENCRYPT_MODE, keyObject);
                String encrypted = new
                    String(cipher.doFinal(message.getBytes()));
                return encrypted;
        } catch (Exception e) {
                return null;
        }
}
```

Listing 4: Encrypting a message in RSA.

So the group can now encrypt its symmetric key, but on the other end the user must be able to decrypt it. The user already contains its own private key object as an attribute, so all that needs to be passed to this function is the message. The user simply decrypts the message using the cipher object as before, this time with the private key. Note that the functionality to decrypt the message in RSA is the same as the functionality to encrypt it, but with the private key instead of the public key. This demonstrates an important property of RSA; that using the public key first, followed by the private key, gives the same result as using the private key first, followed by the public key.

```
protected String decryptRSA(String message) {
        try {
                Cipher cipher = Cipher.getInstance("RSA/ECB/PKCS1Padding");
                cipher.init(Cipher.DECRYPT_MODE, this.privateKey);
                String decryption = new
                    String(cipher.doFinal(message.getBytes()));
                return decryption;
        } catch (Exception e) {
                e.printStackTrace();
                return null;
        }
}
```

Listing 5: Decrypting a message in RSA.

4 Conclusion

Unfortunately, due to timing constraints, I was unable to build the full secure social application as required in the problem statement. However, I had built some important components of the final solution, which would have been fundamental to achieving my goal. After initially spending too much time focusing on the database, I prioritized my objectives and layed out knowledge I wanted to consolidate in this assignment. The problem statement focused on key management, encryption, decryption and addition/removal of people to/from groups. I was able to set out a theoretical approach to solving this problem using concepts taught in lectures. Through a combination of symmetric and asymmetric encryption, all of the desired functionality could be achieved using the model I planned to follow. By exploring the concepts discussed in class before the assignment, I could better understand how to implement this model. As I began to code individual parts, the knowledge I picked up backed up what was learnt in lectures. I saw practically how AES could be put into practise, using concepts such as shared keys, CBC and initialization vectors. I also got to practically explore RSA and write code which could be used to safely transmit a shared key. Though the final deliverable was not completed, I still learned a large amount from this assignment by investigating the theory, modelling the application in my head, and beginning to develop using many of the key concepts that were touched on in class.

5 Appendix

```
import java.io.*;
import java.util.*;
import java.util.logging.Level;
import java.util.logging.Logger;


/*
 *      The aim of this project is to develop a secure social media
    application for Facebook,
 *  Twitter, WhatsApp etc., or for your own social networking app. For
    example, your
 *  application will secure the Facebook Wall, such that only people that are
    part of
 *  your  Secure  Facebook Group  will be able to decrypt each others
    posts. To all other
 *  users of the system the post will appear as ciphertext. You are required
    to design
 *  and implement a suitable key management system for your application that
    allows any
 *  member of the group to share social media messages securely, and allows
    you to add
 *  or remove people from a group. You are free to implement your application
    for a
 *  desktop or mobile platform and make use of any open source cryptographic
    libraries.
 */

public class SecureSocialApp {
        public static void main(String[] args) {
                Logger mongoLogger = Logger.getLogger("org.mongodb.driver");
                mongoLogger.setLevel(Level.SEVERE); // e.g. or Log.WARNING,
                    etc.
                Scanner inputScanner = new Scanner(System.in);
                DBClient mongo = createDBClient();
                System.out.println("Welcome to Secure Social.");
                User currentUser = signIn(inputScanner, mongo);
                Group group = new Group("testgroup");
                String encryption = currentUser.encryptAES("test", group.key);
                System.out.println(encryption);
                String decryption = currentUser.decryptAES(encryption,
                    group.key);
                System.out.println(decryption);
                return;
        }

        /*
         * Create an instance of the signed in user.
         */
        private static User signIn(Scanner inputScanner, DBClient mongo) {
                String userName, password;
                boolean validated = false;
                do {
                        System.out.println("Please enter your username.");
                        userName = inputScanner.nextLine();
```

```
                        System.out.println("Please enter your password.");
                        password = inputScanner.nextLine();
                        if(mongo.validateLogin(userName, password))
                        {
                                validated = true;
                        } else {
                                System.out.println("Invalid login. You will
                                    need to re-enter your credentials.");
                        }
                } while(!validated);
                User user = new User(userName, password);
                return user;
        }

        /*
         * Create a client to interface with MongoDB using saved credentials.
         */
        private static DBClient createDBClient() {
                String[] passwords = getPasswords();
                String readOnlyPassword = passwords[0];
                String readWritePassword = passwords[1];
                return new DBClient("readwrite", readWritePassword);
        }

        /*
         * Return the passwords to the data base in a String array. index 0
            contains the
         * read-only password index 1 contains the read-write password
         */
        private static String[] getPasswords() {
                try {
                        String[] passwords = new String[2];

                        FileReader fileReader = new
                            FileReader("readonlypassword.txt");
                        Scanner fileScanner = new Scanner(fileReader);
                        passwords[0] = fileScanner.nextLine();
                        fileReader.close();
                        fileScanner.close();

                        fileReader = new FileReader("readwritepassword.txt");
                        fileScanner = new Scanner(fileReader);
                        passwords[1] = fileScanner.nextLine();
                        fileReader.close();
                        fileScanner.close();
                        return passwords;
                } catch (Exception e) {
                        e.printStackTrace();
                        return null;
                }
        }
}
```

Listing 6: SecureSocialApp.java, which contains the mainline for the project. This is where the bulk of the integration of concepts would be, including interfacing with the main user. It currently contains code to connect to the database, sign in the current user and run a simple AES test.

```
import java.security.KeyFactory;
import java.security.PublicKey;
import java.security.spec.X509EncodedKeySpec;
import java.util.Base64;
import javax.crypto.Cipher;
import javax.crypto.KeyGenerator;
import javax.crypto.SecretKey;

public class Group {

        protected String groupname;
        SecretKey key;

        public Group(String groupname) {
                try {
                        // Initialise the AES-256 Key Generator
                        KeyGenerator keyGenerator =
                            KeyGenerator.getInstance("AES");
                keyGenerator.init(256);
                // Generate the Key
                SecretKey key = keyGenerator.generateKey();
                this.key = key;
                } catch (Exception e) {
                        e.printStackTrace();
                }
        }

        /*
         * Take a message and a public key in textual form and encode
         * the message with the key using RSA.
         */
        protected String encryptRSA(String message, String publicKey) {
                try {
                        X509EncodedKeySpec keySpec = new
                            X509EncodedKeySpec(Base64.getDecoder().decode(publicKey.ge
                        PublicKey keyObject =
                            KeyFactory.getInstance("RSA").generatePublic(keySpec);
                        Cipher cipher =
                            Cipher.getInstance("RSA/ECB/PKCS1Padding");
                        cipher.init(Cipher.ENCRYPT_MODE, keyObject);
                        String encrypted = new
                            String(cipher.doFinal(message.getBytes()));
                        return encrypted;
                } catch (Exception e) {
                        return null;
                }
        }
}
```

Listing 7: Group.java, which contains the object representing the group. This is where all of the functionality of the group as a concept would be. Currently contains creation of a symmetric key for the group, and RSA encryption using a user's public key.

```
import java.security.Key;
import java.security.KeyPair;
import java.security.KeyPairGenerator;
import java.util.Map;
import javax.crypto.Cipher;
import javax.crypto.SecretKey;
import javax.crypto.spec.IvParameterSpec;
import javax.crypto.spec.SecretKeySpec;

public class User {

        protected String username;
        protected String password;
        protected Key privateKey;
        protected Key publicKey;
        protected boolean admin;

        public User(String username, String password) {
                try {
                        this.username = username;
                        this.password = password;
                        this.admin = false;
                        KeyPairGenerator keyPairGenerator;
                        keyPairGenerator =
                            KeyPairGenerator.getInstance("RSA");
                        keyPairGenerator.initialize(2048);
                        KeyPair kp = keyPairGenerator.genKeyPair();
                        this.publicKey = kp.getPublic();
                        this.privateKey = kp.getPrivate();
                } catch (Exception e) {
                        e.printStackTrace();
                }
        }

        /*
         * Decrypt a message with RSA using your private key.
         */
        protected String decryptRSA(String message) {
                try {
                        Cipher cipher =
                            Cipher.getInstance("RSA/ECB/PKCS1Padding");
                        cipher.init(Cipher.DECRYPT_MODE, this.privateKey);
                        String decryption = new
                            String(cipher.doFinal(message.getBytes()));
                        return decryption;
                } catch (Exception e) {
                        e.printStackTrace();
                        return null;
                }
        }
```

```
/*
 * Encrypt a message with AES using a secret key. Returns a string
    which is the
 * initialization vector appended to the encoded text.
 */
protected String encryptAES(String message, SecretKey key) {
        try {
                // Convert the message to bytes for us to work with
                byte[] plainText = message.getBytes();
                // Initialize the cipher using the secret key
                SecretKeySpec keySpec = new
                    SecretKeySpec(key.getEncoded(), "AES");
                Cipher cipher =
                    Cipher.getInstance("AES/CBC/PKCS5Padding");
                cipher.init(Cipher.ENCRYPT_MODE, keySpec);
                // Encrypt the message
                byte[] encrypted = cipher.doFinal(plainText);
                // Append the initialisation vector to the start of
                    the payload
                byte[] iv = cipher.getIV();
                byte[] payload = new byte[iv.length +
                    encrypted.length];
                System.arraycopy(iv, 0, payload, 0, iv.length);
                System.arraycopy(encrypted, 0, payload, iv.length,
                    encrypted.length);
                // Return the payload
                String cipherText = new String(payload);
                return cipherText;
        } catch (Exception e) {
                e.printStackTrace();
                return null;
        }

}

protected String decryptAES(String message, SecretKey key) {
        try {
                // Convert the message to bytes for us to work with
                byte[] payload = message.getBytes();
                // Take off the initialization vector which was used
                    to encrypt
                byte[] iv = new byte[16];
                byte[] cipherText = new byte[payload.length -
                    iv.length];
                System.arraycopy(payload, 0, iv, 0, 16);
                System.arraycopy(payload, iv.length, cipherText, 0,
                    cipherText.length);
                // Decrypt the message now that we know the
                    initialization vector
                SecretKeySpec keySpec = new
                    SecretKeySpec(key.getEncoded(), "AES");
                Cipher cipher =
                    Cipher.getInstance("AES/CBC/PKCS5Padding");
                cipher.init(Cipher.DECRYPT_MODE, keySpec, new
```

```
                        IvParameterSpec(iv));
                    // Convert to the string form that we want and return
                    String decryptedText = new
                        String(cipher.doFinal(cipherText));
                    return decryptedText;
                } catch (Exception e) {
                    e.printStackTrace();
                    return null;
                }
        }

        public boolean isAdmin() {
                return this.admin;
        }

        public void writePost() {

        }

        public String getUsername() {
                return this.username;
        }
}
```

Listing 8: User.java, the objects to invoke all of the functionality of a user. Currently contains functionality to encrypt & decrypt through AES, generate a pair of RSA keys, and decode a message using its own RSA key.

```
import java.lang.*;
import java.io.*;
import java.util.*;
import java.util.logging.Level;
import java.util.logging.Logger;
import org.bson.Document;
import com.mongodb.BasicDBObject;
import com.mongodb.ConnectionString;
import com.mongodb.client.MongoClients;
import com.mongodb.client.MongoCollection;
import com.mongodb.client.MongoClient;
import com.mongodb.client.MongoDatabase;
import com.mongodb.MongoClientSettings;
import com.mongodb.client.model.Filters;
import com.mongodb.client.model.Updates;

/*
 *      Class used to interface with the Mongo database.
 */

public class DBClient {
        private MongoClient mongoClient;
        private MongoDatabase database;
        private MongoCollection<Document> users;
        private MongoCollection<Document> groups;
        private MongoCollection<Document> posts;

        DBClient(String name, String password) {
```

```
            this.mongoClient = MongoClients.create("mongodb+srv://" +
                name + ":" + password
                        +
                            "@secure-social-0n8uh.azure.mongodb.net/test?retry
            this.database = mongoClient.getDatabase("Secure-Social");
            this.users = database.getCollection("Users");
            this.groups = database.getCollection("Groups");
            this.posts = database.getCollection("Posts");
    }

    /*
     * Check that a login matches the username and password stored in the
         db.
     */
    public boolean validateLogin(String userName, String password) {
            Document user = users.find(Filters.and(Filters.eq("username",
                userName), Filters.eq("password", password)))
                            .first();
            if (user == null) {
                    return false;
            } else {
                    return true;
            }
    }

    /*
     * Create a new entry in the posts document.
     */
    public void createPost(Scanner inputScanner, String userName) {
            System.out.println("Please␣enter␣the␣text␣entry␣you␣would␣
                like␣to␣post:␣");
            String post = inputScanner.nextLine();
            Date date = new Date();
            Document document = new Document("username",
                userName).append("date", date).append("content", post);
            posts.insertOne(document);
            System.out.println("Post␣created␣successfully.");
    }

    /*
     * Create a new entry in the users document.
     */
    public void createUser(Scanner inputScanner) {
            System.out.println("Please␣enter␣the␣username␣of␣the␣user␣to␣
                create:␣");
            String userName = inputScanner.nextLine();
            System.out.println("Please␣enter␣the␣password␣of␣the␣user␣to␣
                create:␣");
            String userPassword = inputScanner.nextLine();
            Document document = new Document("username",
                userName).append("password", userPassword);
            users.insertOne(document);
            System.out.println("User␣created␣successfully.");
    }
```

```
        /*
         * Create a new entry in the groups document.
         */
        public void createGroup(Scanner inputScanner) {
                System.out.println("Please enter the name of the group to
                    create: ");
                String groupName = inputScanner.nextLine();
                Document document = new Document("groupname",
                    groupName).append("users", Arrays.asList());
                groups.insertOne(document);
                System.out.println("Group created successfully");
        }

        /*
         * Add a user to the groups document, provided the user and group
             exist.
         */
        public void addToGroup(Scanner inputScanner) {
                System.out.println("Please enter the name of the group to add
                    a member to: ");
                String groupName = inputScanner.nextLine();

                if (!isGroup(groupName)) {
                        System.out.println("This group does not exist");
                } else {
                        System.out.println("Please enter the username of the
                            user that you want to add to the group");
                        String userName = inputScanner.nextLine();
                        if (!isUser(userName)) {
                                System.out.println("This user does not
                                    exist.");
                        } else {
                                groups.updateOne(Filters.eq("groupname",
                                    groupName), Updates.addToSet("users",
                                    userName));
                                System.out.println("User added
                                    successfully.");
                        }
                }
        }

        /*
         * Remove a user from the groups document, provided the user and
             group exist.
         */
        public void removeFromGroup(Scanner inputScanner) {
                System.out.println("Please enter the name of the group to
                    remove a member from: ");
                String groupName = inputScanner.nextLine();

                if (!isGroup(groupName)) {
                        System.out.println("This group does not exist");
                } else {
                        System.out.println("Please enter the username of the
                            user that you want to remove to the group");
```

```
                String userName = inputScanner.nextLine();
                if (!isUser(userName)) {
                        System.out.println("This user does not
                            exist.");
                } else {
                        groups.updateOne(Filters.eq("groupname",
                            groupName), Updates.pull("users",
                            userName));
                        System.out.println("User removed
                            successfully.");
                }
        }
}

/*
 * Check if the specified user is in the group in the db.
 */
public boolean isInGroup(String userName, String groupName) {
        ArrayList<String> users = (ArrayList<String>)
            groups.find(Filters.eq("groupname",
            groupName)).first().get("users");
        if(users.contains(userName)) {
                return true;
        } else {
                return false;
        }
}

/*
 * Check if the specified group exists in the db.
 */
public boolean isGroup(String groupName) {
        return (groups.find(Filters.eq("groupname",
            groupName)).first() != null);
}

/*
 * Check if the specified user exists in the db.
 */
public boolean isUser(String userName) {
        return (users.find(Filters.eq("username", userName)).first()
            != null);
}

private MongoCollection<Document> getUsers() {
        return this.users;
}

private MongoCollection<Document> getGroups() {
        return this.users;
}

private MongoCollection<Document> getPosts() {
        return this.posts;
}
```

```
}
```

Listing 9: User.java, which was created to interface with the MongoDB originally intended to hold all of the data for this project. Unfortunately too much time was spent developing this, but it does contain a lot of functionality that would be needed in the final product.