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

CSU34031 Advanced Telecommunications Securing Social Media Applications

Kamil Przepiórowski
17327895

April 14, 2020

Contents

1 Requirements

• Develop a secure social media application

• Secure a Group/Wall such that one people part of it will be able to decrypt each other's posts

• Users outside the Group/Wall will see the posts as ciphertext

• Design and implement a suitable key management system

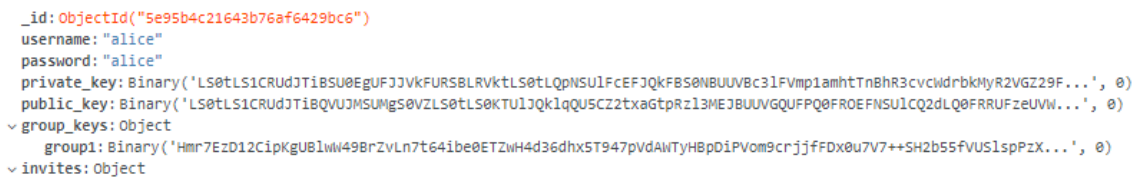
• Add & remove users from the group

2 Overview of design

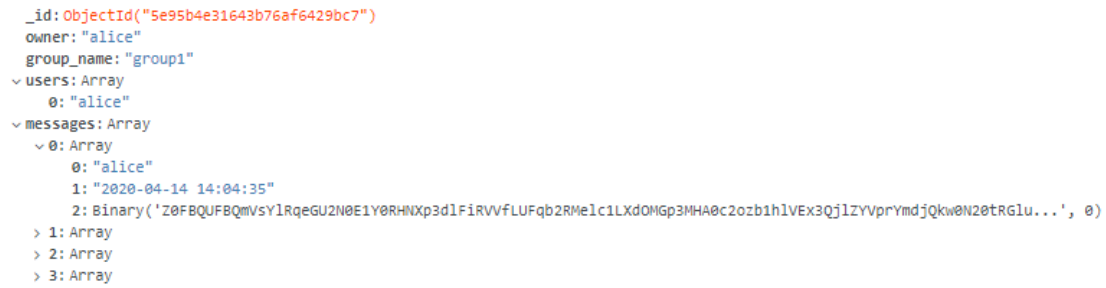
For this project I decided not to use an existing social media like Facebook or Twitter, but rather make a simulation of my own. This was because I wanted to focus more on experimenting with the crypto side of things rather than dealing with APIs etc, and thought that making my own 'social media' would give me more of a sandbox environment without limitations. This is also why I opted for a command line interface rather than developing some frontend using JS.

Overall my design is this: RSA with AES on top, but without Certs. (more on this in the next section). I ended up storing everything in a MongoDB just for simplicity, though of course in a real system you wouldn't store plaintext passwords and private keys in a DB like that.

A user object in my DB has the following attributes:



A group object has the following attributes:



You can find a video demo highlighting the functionality of my application here : https://youtu.be/jOjG3xnEIiI

3 Key Management Infrastructure

As mentioned, I use a hybrid scheme with RSA and AES but without Certs in my implementation. Upon registration, a user generates their private and public keys. I use 2048 bit keys. To do this I use the RSA Python library https://pycryptodome.readthedocs.io/en/latest/src/public_key/rsa.html.

For the groups, I decided to have one group key per group which is used to encrypt/decrypt the posts inside the group. I considered having a new 'session key' for each message, but decided against it as I wanted new users to see past posts in the group as well, before their arrival. To generate the symmetric key I used the Python library Fernet https://cryptography.io/en/latest/fernet/. Fernet uses AES in CBC mode with a 128-bit key for encryption; using PKCS7 padding. IVs are generated using os.urandom().

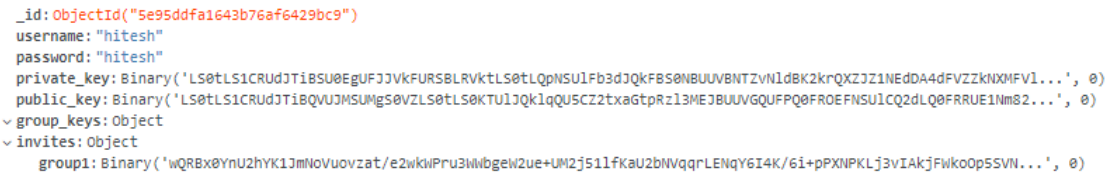
Users store a list of groups which they are part of. For each group that they are part of, they store a dictionary entry of of group name as key and the group key encrypted with the users public key as value. The group object in the DB does not store the group key itself. The owner adds key to his list when creating group and then can invite other users by decrypting the group key with their own private key, encrypting it with the target's public key and sending it to them. As I don't use Certs, this is a potential weakness in the design as there is no guarantee that the message received is from the original sender and that the message is untampered. Overall I also put quite a lot of trust into the connections to the DB to be secure, which again could be a point of weakness.

4 Encryption and Decryption of Posts

As shown in the demo video, users who are part of the group can see the group posts in their decrypted form, and users outside of the group see the messages as ciphertext. The group object in the DB stores the messages encrypted with the group key. The group key is a symmetric key and is used for both encryption and decryption of the messages.

5 Group Membership

As mentioned above and mentioned in the video - only the owner of the group has the right to kick people from the group, but each member of the group can invite other existing users into the group. The source of the invite decrypts the group key with their private key, encrypts it with the public key of the target and sends the invitation. The way I simulate sending invitations is just updating the invite attribute of the target user in the DB. For example, if Alice wants to invite Hitesh into group1, the following happens:



Hitesh has no group keys yet as he is a newly registered user, but he has an invitation to group1.

As shown in the video, a user can check their inbox/invitations and then choose to either join the group, or clear their inbox, which equates to rejection of the invite.

The owner kicks people by simply removing the group entry from the targets group keys dictionary. The group key itself is constant throughout the existence of the group, which again could be a point of weakness as someone who was previously in the group but got kicked could potentially leak the key if they retained access to it somehow. This could possibly be solved by key renegotiation / regeneration whenever a user gets kicked / joins the group, however I considered this to be out of scope of my application.

6 Code

The code here is very badly formatted, please see my GitHub repo for a good version : https://github.com/kamilprz/SecureSocialMedia

```
import datetime
import Crypto
from Crypto.PublicKey import RSA
from Crypto.Cipher import PKCS1_OAEP
from cryptography.fernet import Fernet
import base64
from pymongo import MongoClient
from getpass import getpass
from tmp import temp


mongoC = temp.getMongoClient()
client = MongoClient(mongoC)
db = client.get_database('telecomms_db')
users = db.users
groups = db.groups

# navigation in the program - since there is no GUI
def main():
    try:
        # loginUser represents the currently logged in user
        # if loginUser = None, no user currently logged in
        # otherwise it has the value of the currently logged in user object
        loginUser = None
        while True:
            action = input('\nWhat would you like to do >>> ')
            action = action.split(' ')
            if action[0] == 'help':
                print_help()
                continue

            # not logged in
            if loginUser is None:
                if action[0] == 'register':
                    register()

                elif action[0] == 'login':
                    loginUser = login(loginUser)

                else:
                    print('Invalid input. Type \'help\' for more info.')

            # logged in
            else:
```

```
                if action[0] == 'create':
                    # create <group>
                    create_group(action[1], loginUser)

                elif action[0] == 'post':
                    # post <group>
                    post_to_group(action[1], loginUser)

                elif action[0] == 'view':
                    # post <group>
                    view_group(action[1], loginUser)

                elif action[0] == 'invite':
                    # invite <user> <group>
                    invite(action[1], action[2], loginUser)

                elif action[0] == 'kick':
                    # kick <user> <group>
                    kick(action[1], action[2], loginUser)

                elif action[0] == 'join':
                    # join <group>
                    loginUser = join_group(action[1], loginUser)

                elif action[0] == 'inbox':
                    # inbox
                    loginUser = view_inbox(loginUser)

                elif action[0] == 'clear':
                    # clear
                    clear_inbox(loginUser)

                elif action[0] == 'logout' or action[0] == 'stop':
                    # logout
                    loginUser = logout(loginUser)

                else:
                    print('Invalid input. Type \'help\' for more info.')

    except (KeyboardInterrupt, SystemExit):
        print('\n\nShutting down...')


# prints all the available commands that a user can enter
def print_help():
    print("""
    When not logged in:
        login
        register
```

```
        help

    When logged in:
        create <group>
        post <group>
        view <group>
        invite <user> <group>
        kick <user> <group> (owner only)
        join <group>
        inbox
        clear
        logout
    """)


# registers a new user into the database
def register():
    print('Please enter a username and password.')
    username = input('Username: ')
    password = getpass()
    private_key, public_key = generate_keys()
    new_user = {
        'username' : username,
        'password' : password,
        'private_key': private_key.exportKey(),
        'public_key': public_key.exportKey(),
        'group_keys': {},
        'invites': {}
    }
    users.insert_one(new_user)
    print('Registered user: {0}'.format(username))


# generates user's private and public keys
# returns these keys
def generate_keys():
   private_key = RSA.generate(2048)
   public_key = private_key.publickey()
   return private_key, public_key


# encrypts a group key using the users public_key
# returns encrypted_key
def encrypt_group_key(user, group_key):
    public_key = user['public_key']
    public_key = RSA.importKey(public_key)
    cipher = PKCS1_OAEP.new(key = public_key)
    encrypted_key = cipher.encrypt(group_key)
    return encrypted_key
```

```
# decrypts the encrypted_key of group_name using user's private_key
# returns group_key
def decrypt_group_key(user, group_name):
    encrypted_key = user['group_keys'][group_name]
    private_key = user['private_key']
    private_key = RSA.importKey(private_key)
    decrypt = PKCS1_OAEP.new(key = private_key)
    group_key = decrypt.decrypt(encrypted_key)
    return group_key


# log the user in
# returns a new loginUser object
def login(loginUser):
    username = input('Username: ')
    user = users.find_one({'username': username})
    if user:
        password = getpass()
        if password == user['password']:
            loginUser = user
            print('Logged in as: {0}'.format(loginUser['username']))
        else:
            print('Wrong password.')
    else:
        print('User not found.')
    return loginUser


# logs the user out
# returns loginUser as None to simulate logout
def logout(user):
    user = None
    print('Logged out.')
    return user


# a logged in user can create a group
def create_group(group_name, owner):
    # owner is the loginUser who called create_group()
    owner_username = owner['username']

    # generate a symmetric key for the group
    group_key = Fernet.generate_key()
    f = Fernet(group_key)

    # group creation message is posted into the group, with current datetime a
    dt = datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S")
```

```
    message = 'Created {0}. Hello, world!'.format(group_name)
    token = f.encrypt(message.encode())
    new_group = {
        'owner': owner_username,
        'group_name': group_name,
        'users': [owner_username],
        'messages': [(owner_username, dt, token)]
    }
    groups.insert_one(new_group)
    print('Created group: {0}'.format(group_name))

    # encrypt group_key with owners public_key and add it to their group_keys
    encrypted_key = encrypt_group_key(owner, group_key)
    group_keys = owner['group_keys']
    group_keys.update({group_name: encrypted_key})
    owner_updates = {
        'group_keys': group_keys
    }
    users.update_one({'username': owner_username}, {'$set': owner_updates})


# a logged in user who is part of the group can post a message to the group
# a user not part of the group gets an error message
def post_to_group(group_name, user):
    group = groups.find_one({'group_name': group_name})
    if group:
        if check_membership(user, group):
            message = input('Post to {0}:\n'.format(group_name))
            group_key = decrypt_group_key(user, group_name)

            # encrypt the message using the group_key
            f = Fernet(group_key)
            token = f.encrypt(message.encode())
            dt = datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S")
            messages = group['messages']
            messages.append((user['username'], dt ,token))
            group_updates = {
                'messages': messages
            }
            groups.update_one({'group_name': group_name}, {'$set': group_updat
        else:
            print('You must be part of the group to post to it.')
    else:
        print('This group does not exist.')


# a logged in user can view the group's messages
# author and datetime are left unencrypted to help with visibility
def view_group(group_name, user):
```

```
    group = groups.find_one({'group_name': group_name})
    messages = group['messages']

    # a user not part of the group sees the encrypted messages
    if user['username'] not in group['users']:
        print('\n>>>> Welcome to {0} <<<<'.format(group_name))
        for x in messages:
            print('>>> {0} @ {1}'.format(x[0], x[1]))
            print(x[2].decode() + '\n')

    # a user inside the group is able to decrypt the messages
    else:
        group_key = decrypt_group_key(user, group_name)
        f = Fernet(group_key)
        messages = group['messages']
        print('\n>>>> Welcome to {0} <<<<'.format(group_name))
        for x in messages:
            print('>>> {0} @ {1}'.format(x[0], x[1]))
            print((f.decrypt(x[2])).decode() + '\n')


# a member of a group can invite another existing user to the group
def invite(username, group_name, source):
    group = groups.find_one({'group_name': group_name})
    if group:
        if check_membership(source, group):
            group_key = decrypt_group_key(source, group_name)

            target = users.find_one({'username': username})
            if target:
                if target['username'] == source['username']:
                    print('You cannot invite yourself to a group.')
                else:
                    invite_key = encrypt_group_key(target, group_key)
                    invites = target['invites']
                    invites.update({group_name: invite_key})
                    target_updates = {
                        'invites': invites
                    }
                    users.update_one({'username': target['username']}, {'$set'
                    print('Invite to \'{0}\' has been sent to \'{1}\''.format(
            else:
                print('User \'{0}\' does not exist.'.format(username))
        else:
            print('You must be part of the group to invite to it.')
    else:
        print('This group does not exist.')
```

```
# the owner of a group is able to kick users out of the group
def kick(username, group_name, source):
    group = groups.find_one({'group_name': group_name})
    if group:
        # check if owner
        if source['username'] == group['owner']:
            if source['username'] == username:
                print('You cannot kick yourself from the group.')
            else:
                # delete target's group_key
                target = users.find_one({'username': username})
                if target:
                    if check_membership(target, group):
                        user_groups = target['group_keys']
                        try:
                            del user_groups[group_name]
                        except KeyError:
                            pass
                        user_update = {
                            'group_keys': user_groups
                        }
                        users.update_one({'username': target['username']}, {'$
                        print('\'{0}\' has been kicked from \'{1}\'.'.format(t

                        # post a message to the group that user has been kicke
                        group_key = decrypt_group_key(source, group_name)
                        f = Fernet(group_key)
                        dt = datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%
                        message = '\'{0}\' has been kicked from \'{1}\'.'.form
                        token = f.encrypt(message.encode())
                        messages = group['messages']
                        messages.append((source['username'], dt, token))

                        # delete user from group's users
                        group_users = group['users']
                        group_users.remove(username)
                        group_updates = {
                            'users': group_users,
                            'messages': messages
                        }
                        groups.update_one({'group_name': group_name}, {'$set':
                    else:
                        print('User \'{0}\' is not part of this group.'.format
                else:
                    print('User \'{0}\' does not exist.'.format(username))
        else:
            print('You cannot do that as you\'re not the owner of {0}.'.format
    else:
        print('This group does not exist.')
```

```
# a logged in user can check their inbox - shows if they have any invitations
def view_inbox(user):
    updated_user = users.find_one({'username': user['username']})
    if updated_user['invites']:
        print('You are invited to {0} group(s).'.format(len(updated_user['invi
        for x in updated_user['invites']:
            print('>> {0}'.format(x))
    else:
        print('Your inbox is empty.')
    return updated_user


# a logged in user can clear their inbox - removing any exsiting invitations
def clear_inbox(user):
    invites = {}
    invites_update = {
        'invites': invites
    }
    users.update_one({'username': user['username']}, {'$set': invites_update})
    print('Cleared inbox for \'{0}\''.format(user['username']))


# if have any invitations to groups, can join the group
# returns an updated object of the user
def join_group(group_name, user):
    group = groups.find_one({'group_name': group_name})
    if group:
        user = users.find_one({'username': user['username']})
        try:
            # add group_name and encrypted_key to group_keys and delete invite
            invite_key = user['invites'][group_name]
            invites = user['invites']
            try:
                del invites[group_name]
            except KeyError:
                print('Unable to delete the invite.')
            user_groups = user['group_keys']
            user_groups.update({group_name: invite_key})
            user_update = {
                'group_keys': user_groups,
                'invites': invites
            }
            users.update_one({'username': user['username']}, {'$set': user_upd

            # decrypt invite_key into group_key to encrypt join message
            group_key = decrypt_group_key(user, group_name)
            f = Fernet(group_key)
```

```
            dt = datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S")
            message = '\'{0}\' has joined the group. Welcome!'.format(user['us
            token = f.encrypt(message.encode())
            messages = group['messages']
            messages.append((user['username'], dt, token))

            # add username to groups usernames, and post a join message
            group_users = group['users']
            group_users.append(user['username'])
            group_updates = {
                'users': group_users,
                'messages': messages
            }
            groups.update_one({'group_name': group_name}, {'$set': group_updat
            print('You have successfully joined {0}'.format(group_name))
        except KeyError:
            print('You do not hold an invitation to this group.')
            return user
    else:
        print('This group does not exist.')
    return user


# checks whether a user is part of given group
def check_membership(user, group):
    if user['username'] in group['users']:
        return True
    return False


if __name__ == '__main__':
    main()
```