CSU34031 Advanced Telecommunications
Project II

Efeosa Eguavoen - 17324649

April 11, 2020

1 Project Specificiation

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 other's 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.

2 Overall Implementation

To implement this project, I went with creating my own social media website using the Django framework for Python. I will split this report into 2 major sections:

1. Views: The backend of the code and how things connect. A diagram of some description will be added also.
2. Models: Databases, Html code and other stuff used to implement the project.

3 Views

The project was made using Django, a web framework for Python that has a lot of built in features that makes it easier to get a website built and running quickly. The project is composed of 2 main applications:

1. Accounts : An app that manages user creation, login and user authentication. Also manages user profiles.
2. SecureApp: An app that manages user posts, creating connections between users and encryption.

3.1 Accounts

The accounts app is responsible for creating users for the social media application and logging them in. It also handles user account verification and their user profiles. On launch of the project, we first have to either login or register. Logging in is handled by Django using their authentication library that handles users and passwords securely by encrypting them and storing them in the system. The login form I've created verifies input and makes sure that nothing that shouldn't be entered is entered.Different error messages are returned depending on the type of error the user makes as defined in the forms.py file in accounts.

```
def login_view(request):
    next = request.GET.get('next')
    form = UserLoginForm(request.POST or None)
    if form.is_valid():
        username = form.cleaned_data.get('username')
        password = form.cleaned_data.get('password')
        user = authenticate(username=username, password=password)
        login(request, user)
        if next:
            return redirect(next)
        return redirect('home')
    context = {
        'form': form,
    }
    return render(request, "clogin.html", context)
```

Account registration is handled in another function that asks the user for multiple pieces of data to register a new user to the system. Passwords are required to be of certain length and containing certain characters for security. Passwords are also asked to be entered twice for added security and to make sure the user isn't creating spam accounts. Different error messages are returned depending on the type of error the user makes as defined in the forms.py file in accounts.

```
def register(request):
    if request.method =='POST':
        form = RegistrationForm(request.POST)
        if form.is_valid():
            form.save()
            return redirect(reverse('home'))
    else:
        form = RegistrationForm()
    args = {'form': form}
    return render(request, 'reg_form.html', args)
```

Account profiles are handled in this app also. Depending on wheter you're viewing your own profile or another profile, different things happen. If you're viewing your own profile, all you details and posts come up unencrypted as it's your own data you're

looking at.
But if you're looking at another users profile, the system first checks if you're friends with the person. If you are, you can see their details and all their posts unencrypted. Also no option to add them as a friend is put up but an option to remove them as a friend is disaplayed.
If you are not friends, all their posts are encrypted using the cryptography library in python. A key is generated and using this key, every post by that user is encrypted and cipher text is instead returned. You then have the option to add them as a friend to unencrypt their posts. But their posts will not be unencrypted until they also add you as a friend.

```
def view_profile(request, pk=None):
    k = Fernet.generate_key()
    crypter = Fernet(k)
    if pk:
        user = User.objects.get(pk=pk)
        friend = userFriends.objects.get(current_user=request.user)
        friends = friend.users.all()
        isFriend = friends.filter(id=user.id).exists()
        userPosts = Post.objects.filter(user_id=user.id)
        if(not isFriend):
            for i in userPosts:
                encrypted = crypter.encrypt(i.post.encode())
                i.post = encrypted
    else:
        user = request.user
    args = {'user': user,'posts':userPosts,"friend":isFriend}
    return render(request, 'profile.html', args)
```

3.2 SecureApp

The actual social media aspect of the app sends you to a homescreen once you've logged. Here you can see posts from everyone on the app. But only posts from people you're friends with and are also friends with you are decrypted. Everyone else's text is ciphertext.
We have the option to submit posts to the site for other users of your group to read. Members who are not part of your group will only see ciphertext

```
class HomeView(TemplateView):
    template_name = 'home.html'

    def get(self, request):
        k = Fernet.generate_key()
        crypter = Fernet(k)
        form = HomeForm()
        posts = Post.objects.all().order_by('-created')
        users = User.objects.exclude(id=request.user.id)
        friend = userFriends.objects.filter(current_user=request.user)
        friends = None
        if(len(friend) > 0):
            friends = userFriends.objects.get(current_user=request.user).users.all()
        for i in posts:
            otherUser = i.user
            otherUserlist = userFriends.objects.filter(current_user=otherUser)
            if(len(otherUserlist) > 0):
                ousers = userFriends.objects.get(current_user=otherUser).users.all().filter(id=request.user.id).exists()
            print(otherUserlist)
            if(not i.user.id == request.user.id):
                if friends is None or not friends.filter(id=i.user.id).exists() or not ousers:
                    encrypted = crypter.encrypt(i.post.encode())
                    i.post = encrypted
        args = {
            'form': form, 'posts': posts, 'users': users, 'friends': friends
        }
        return render(request, self.template_name, args)

    def post(self, request):
        form = HomeForm(request.POST)
        if form.is_valid():
            post = form.save(commit=False)
            post.user = request.user
            post.save()

            text = form.cleaned_data['post']
            form = HomeForm()
            return redirect('home')

        args = {'form': form}
        return render(request, self.template_name, args)
```

Finally we can also add a friend, but as previously mentioned, their posts remain encrypted till they also add you as a friend. You can also see all your friends on this page and have the option to remove friends if you wish. You can click on their names to look at their profiles if you wish

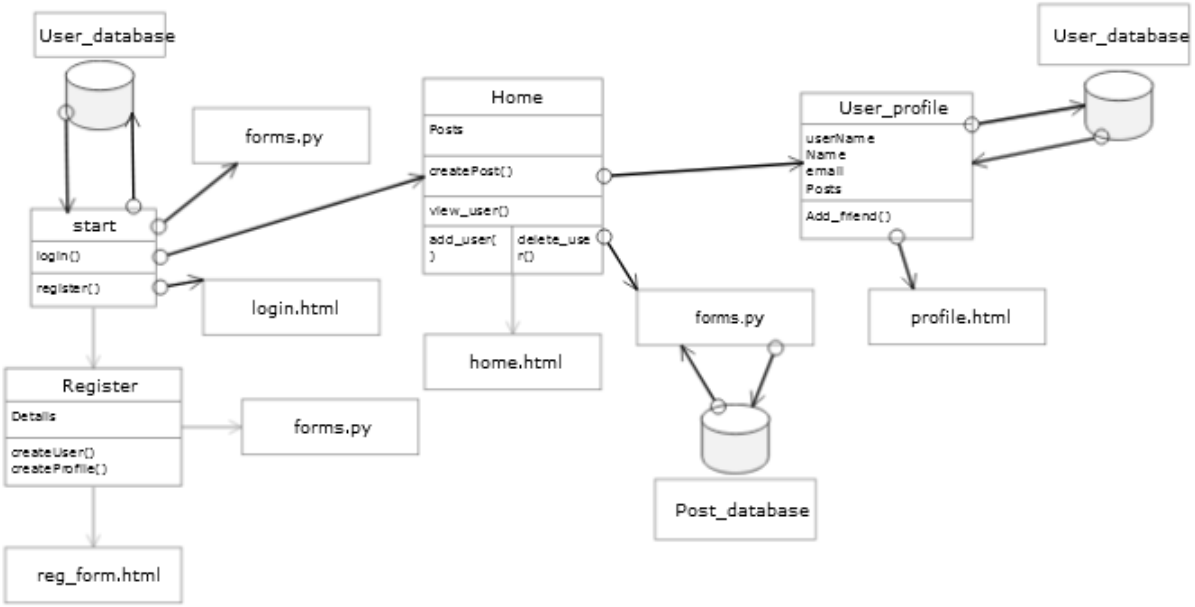
```python
def change_friends(request, operation, pk):
    friend = User.objects.get(pk=pk)
    if operation == 'add':
        userFriends.make_friend(request.user, friend)
    elif operation == 'remove':
        userFriends.lose_friend(request.user, friend)
    return redirect('home')

def change_friendsr(request, operation, pk):
    friend = User.objects.get(pk=pk)
    if operation == 'add':
        userFriends.make_friend(request.user, friend)
    elif operation == 'remove':
        userFriends.lose_friend(request.user, friend)
    return redirect('view_profile_with_pk')
```

3.3 Encryption

For encryption in this app, I used the Cryptography library for python that's the default library used for encryption. The specific function used is called Fernet. It's an implementation of symmetric key authenticated cryptography. It uses AES with cipher block chaining as it's cryptographically strong as suitable for block ciphers. It creates a 128 bit key for encyption. For authentication, it uses HMAC, a cryptograohic hash function that's implemented using SHA 256 as it creates near unique signature for text.

3.4 Diagram



4 Models

A number of other parts are necessary for the project to work. I used SQLite to manage users, posts and relationships between users. Forms were created to take in input from users and enter it into the application. Numerous HTML pages we're also created for the UI and such.

5 Demo

The demo can be located at the following Github link alongside the code:
https://github.com/foesa/TweetSecure
In the attached Demo, I've shown a demonstration of the app working. You can see me create a new user and look at a post made by another user and it's encrypted. When I add the user it remains encrypted but they're added to my friends list. When I logout and login as the other user and add the user that requested, and then log back in as the other user, the post is then decrypted. If I remove the person from my group, their post becomes encrypted again. But once I add them back I can see their post as they didn't remove the current user as a friend.
To run this yourself, clone the project and provided you have python installed, CD into

TweetSecure/SocialMediaApp and then run manage.py. Otherwise, the attached Demo shows all functionality.