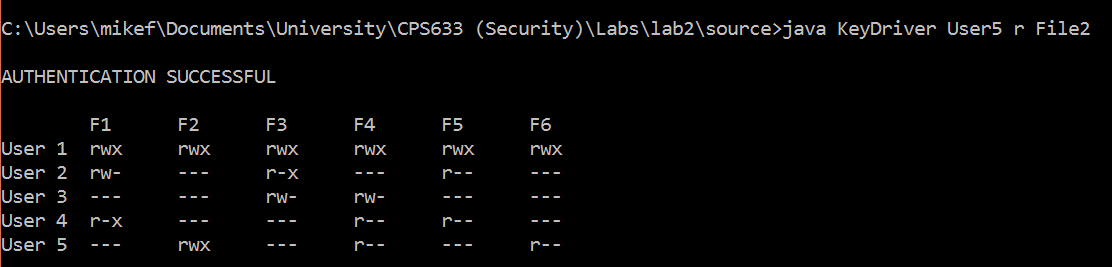
1. **Authentication module.**

In this part of the lab we were supposed to implement the authentication based on keystroke dynamics of the user.

Using provided formula, we calculated deviations and FAR and FRR values, to see if user can be granted authentication.

In this particular instance, our user is always authenticated in order to proceed to further parts of the lab. So, if the program is run as described in part 2.a in the README.txt file, the output depicted in Figure 1 will occur.

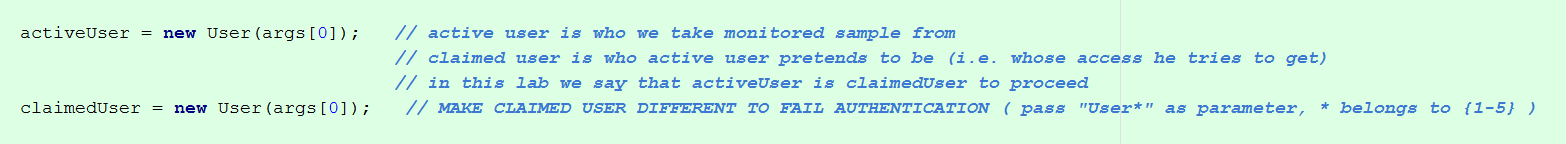


**Figure 1.** Authentication successfully completed and the authorization module takes its turn. (The rest of the output is cut out because it is irrelevant at this point)

After that, the program will pass the user name, permissions and file name to authorization module described later in this report.

The Access Control Matrix is displayed so that it’s easier for the marker to see what data is processed by the program.

Although, in this particular lab user is always who they claim to be, the program internally allows authentication failure.



**Figure 2.** This is a part of code in which we internally say that our monitored user is always who they pretend to be.

Changing the second line of code to “ **claimedUser = new User(“User5”);** “ (which means that whoever tries to access the system, tries to access it exactly through User5) and running it with user name being ANYTHING BUT “User5” will give the result showed on Figure 3.

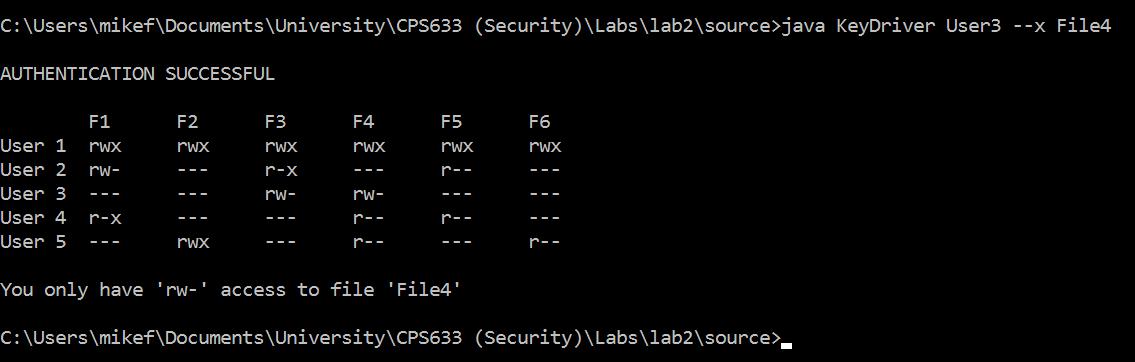


**Figure 3.** Giving the program “User5” vs. giving the program “User3” (considering changes to code made previously)

1. **Authorization module.**

After successful authentication, the program displays the ACM (for marker’s convenience) and then proceeds to authentication part, where it checks if the permissions requested by user match permissions of that user in the ACM. In the first part of Figure 3 we can observe the output in case permissions do match.

If they do not match, the output shown on Figure 4 is produced.



**Figure 4.** User3 does not have “x” permission for File4, so program does not grant access.

Another functionality, required in this lab, is achieved by using the flag –ACL when executing the program. As described in part 2.b of the README.txt file, running program like this will allow user to give a certain user new permissions to a certain file. This is demonstrated on Figure 5.



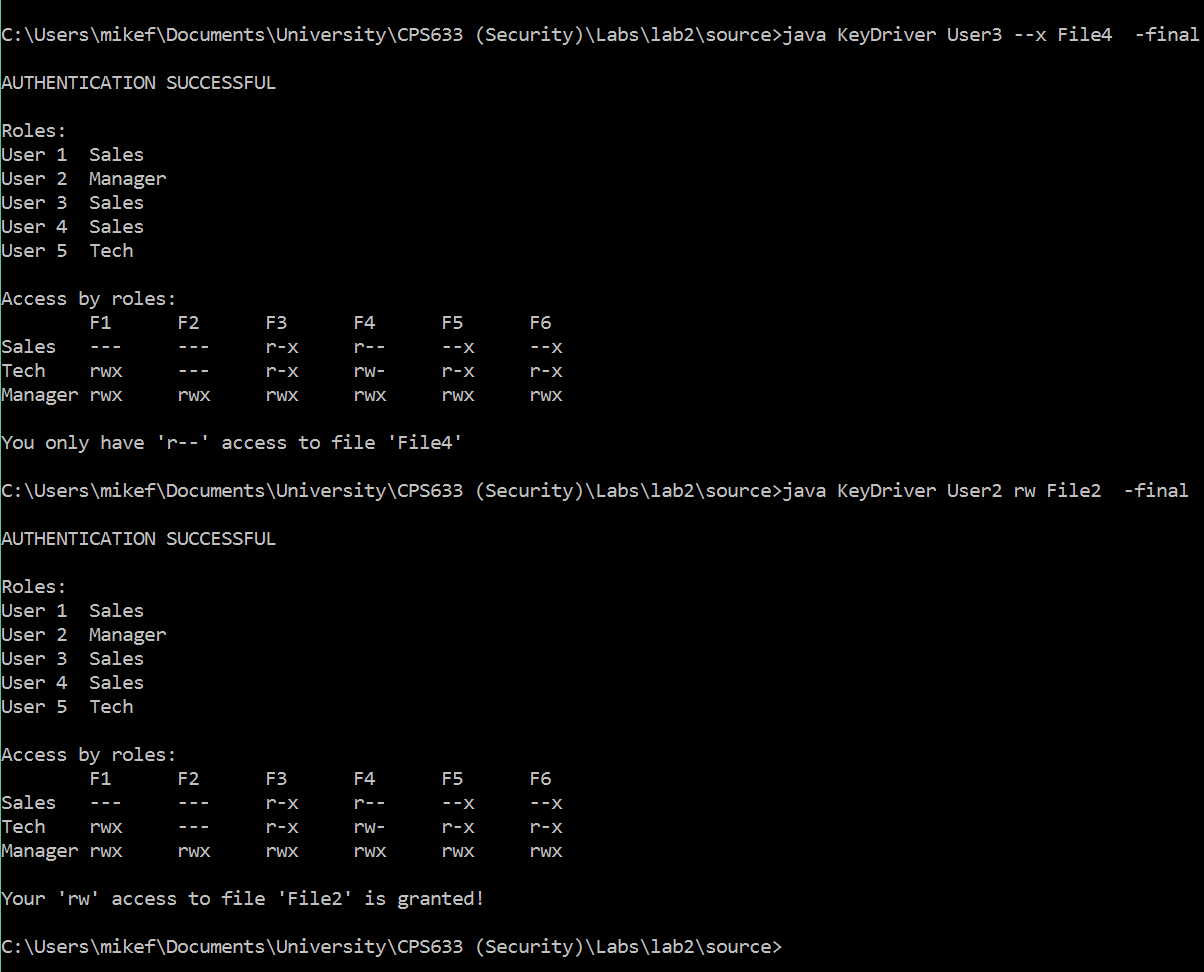
**Figure 5.** Running program with –ACL flag. The program shows the old ACL, updates it with given permissions for given user for given file, displays updated ACL and updates the ACL.txt file. Thus, upon the next execution of the program the permissions will be according to the last update and will not revert back.

1. **Role-Based Access Control**

This part is essentially an extension of the previous part. The only difference is that now the users are not assigned the permissions directly, but rather through different roles: Sales staff, Tech staff and Manager.

In the “source” folder there are files “Roles.txt” and “ACLRole.txt” (described in README.txt) that are used in this part of lab.

When run as described in part 2.c of the README.txt file the output shown on Figure 6 is produced.



**Figure 6.** Running the program with flag –final to perform role based access control check.

In this mode, once the authentication is successful, the program will first display the roles assigned to each user and the ACL showing permissions for each role (for marker’s convenience). After that, it will check what role is assigned to given user, check what permissions that role has for the given file and compare those permissions to the permissions requested by user.

Figure 6 shows the output in both cases, when permissions match (second part) and do not match (first part)