# Pretty Secure File Sharing

*Secured by end-to-end encryption, get absolute control over access to your files*

*Brandon Peh (bcp39), Louise Lee (zl245), Ruixin Ng (rn279), Wang Zilong (zw243)*
*Emails: (netid)@cornell.edu*

## 1. System Purpose

Aim: To provide a secure file sharing system for users.

Users who are logged into the system can create folders and upload files to a private repository. They should be guaranteed that their files are not viewed or modified by others without authorization. Users can share files with other users and choose if the other users are only allowed to download the file, or if they are also given privileges to overwrite and change the permissions of the file. All actions on files and folders are logged in order to ensure accountability.

Users with viewing permissions can view the log associated with a file or folder. Changing the permissions on a folder sets all subfiles and subfolders to have the same permissions along with any other user's permissions set earlier for particular subfiles and subfolders. A concrete example of this: User A has been granted edit rights to file F that is in folder G. When permissions are added to folder G, file F should reflect these new permissions but also retain user A's edit rights (these are *additional*).

A single centralized server will be deployed for clients to connect to. Trusted system administrators are allowed to a log of all user account actions (creation of accounts, change of passwords and account information). They can also delete users and limit the total file size of users. However, they are not allowed to view the file content or filenames of files that users upload or download from the server. This is further explained in our threat model.

All users have to be logged in before performing any actions. We will use a 2FA system (the second factor being email) to log in.

## 2. Sprint Progress

### 2.1 Completed

| Functional Requirements | Importance |
|---|---|
| As an owner, I can upload new files to the system. | M |
| As an owner, I can create new folders in the system. | S |
| As a viewer, I can download files that I have access to view (that have been uploaded by me). | M |
| As a user, I can sign up for a user account. | M |
| As a user/admin, I can change my password. | C |
| As a user/admin, I can change the email associated with my account. . | C |
| As an owner, I can upload new folders in the system. | W |
| As an editor, I can overwrite files that are shared with me. | S |
| As an editor, I can share folders and files that I have uploaded with other users, giving them either read/write access. | S |
| As an editor, I can remove the view/edit privileges of other users. | S |
| As an editor, I can rename folders and files that are shared with me. | C |
| As a viewer, I can download files that I have access to view (that are shared with me by other users). | M |
| As a viewer, I can view the event log of a folder or file that is shared with me. | S |
| As an admin, I can monitor the the logs of user account activity. | S |
| As an admin, I can monitor the the logs of all file activity. | S |
| As an admin, I can monitor the the logs of a specific file activity. | S |
| As an admin, I can remove users from my system. | W |
| As a user, I can delete my account. | W |

### 2.2 System Backlog

| Functional Requirements | Importance |
|---|---|
| As an editor, I can upload new files to folders that are shared with me. | S |

| | |
|---|---|
| As an editor, I can rollback my files to a previous version up to the last 5 days. | W |
| As a user/admin, I can require 2-factor authentication via email in order to log into my account. | W |
| As an admin, I can restrict the total size of files of all users. | C |
| As an admin, I can add users to my system. | W |

# 3. Design Choices

## 3.1 Threat Model

Criminals (Dolev-Yao model)

- Motivation: blackmail, financial gains from sale of sensitive information, steal passwords to access bank accounts
- Capabilities: probably not extensive financial and computational powers, possible social engineering to gain access a system administrator, technical skills to intercept network packets and view/modify these packets, network access to server, no physical access to servers or computers of the user, ability to craft json packets to send to the server outside the framework of our application

Unhappy viewers

- Motivation: curiosity, desire to gain access or know about the presence of files they do not have the privileges to (including if privileges have been revoked), desire to cause trouble by editing files they do not have access to edit
- Capabilities: not much impetus, view access to the files, no physical access to servers or computers of editors, social engineering (could be friends of users with edit rights), ability to edit outgoing json packets by modifying outgoing connections to the server, ability to view their own memory (for old keys)

Server administrators

- Motivation: curiosity, desire to view the contents or file names of files being stored on the server they are running
- Capabilities: ability to see, modify and remove any incoming json packets, ability to see all SQL transactions performed on the database, ability to perform additional SQL transactions outside of our system (granted full access to the database), ability to delete legitimate users and add malicious users to the system
- Their reputation will be damaged and they can be legally prosecuted for deliberate malfeasance if files or user information (such as privileges) are damaged or if the availability of their server is compromised. There is deterrence through accountability because the number of system administrators are expected to be few(less than 10) and it would be easy to find the culprit.
- Therefore we trust the administrators to maintain the integrity of user files and the database, as well as maintain the availability of their server. However, we do not trust the administrators with the confidentiality of the files as it is possible that criminals could gain access as a system administrator.

| | **Threat Model** | **Vulnerability** | **Countermeasure** |
|---|---|---|---|
| 1. | Ability to sniff network | Attacker could view contents of | SSL/TLS ensures that the |

| | | | |
|---|---|---|---|
| | packets. (Criminals) | confidential files. | packets sent are encrypted. |
| 2. | Ability to drop network packets. (Criminals) | Attacker could delete network requests. | SSL/TLS includes a message counter. |
| 3. | Ability to modify network packets. (Criminals) | Attacker could modify network requests. | SSL/TLS includes a signature. |
| | | Attacker could replace the certificate in truststore with a server that they control. | The certificate is signed by our server. |
| 4. | Ability to craft json packets to send to server. (Criminals, unhappy viewers) | Attacker could pretend to be a different user by using the user id of a different user. | Session records the user id of the user the server established the secure channel with. |
| | | Attacker could request for/modify a file or folder that he/she does not have access to. | Authorization is performed on the server side to ensure that the user has the required privilege to access the file. |
| 5. | Ability to sniff memory for old keys. (Unhappy viewers) | Attacker could get the secret key used to encrypt files that he/she had lost access to. | Authorization is performed on server side to ensure that the unauthorized user cannot retrieve the encrypted data. |
| 6. | Ability to gain admin privileges to the database. (Admins, criminals) | Attacker could view the contents/names of confidential files. | Files and names (of files and folders) are encrypted before they are stored on the server. |
| | | Attacker could view the password of users stored in the database. | Passwords are hashed before being stored on the database. |
| | | Attacker could perform a dictionary attack on the list of passwords. | Passwords are salted before being stored on the database. |
| | | Attacker could read the json packet received from users, thereby getting their passwords that are sent during authentication. | Passwords are hashed on client side before being sent in the json packet. |
| | | Attacker could view the private keys of users and thereby decrypt the files. | Password-based encryption of the private key on client side. |

| | | | |
|---|---|---|---|
| | | Attacker could view the secret keys used to encrypt the files/folders. | Secret keys are encrypted using the private keys of viewers of the files/folders. |
| | | Attacker could delete files or users. | Detection through user and file logs. |
| 7. | Strong computational power. (Criminals) | Attacker could perform a brute force online guessing attack on a random accounts. | A strong passwords rule is strictly enforced (basic-16).<br>Rate limiting based on the number of failed logins attempts per session.<br>Rate limiting based on the number of failed login attempts per IP. |
| 8. | Access to a large number of clients interfaces. (Criminals) | Attacker could use numerous clients to perform a brute force online guessing attack attack on an account. | Rate limiting based on the number of failed login attempts to an account. |

### 3.2 Security Goals

1. The system shall prevent the modification of files by unauthorized principals. [I]
2. The system shall prevent the deletion of files by unauthorized principals. [A]
3. The system shall prevent unauthorized principals from viewing the name and content of the files stored on the server. [C]
4. The system shall detect changes made to a folder by a user (creation, deletion) [I]
5. The system shall detect changes made to a file by a user (uploading, deletion, overwriting, renaming) [I]
6. The system shall prevent unauthorized principals from removing the user permissions of a file/folder. [A]
7. The system shall prevent privilege escalation by unauthorized principals. [I]
8. The system shall detect the change of privileges made to all files and folders. [I]
9. The system shall prevent excessive usage of the server's storage capacity. [A]
10. The system shall prevent unauthorized viewing of the previous versions of files. [C]
11. The system shall prevent unauthorized modifications of the previous versions of files. [I]
12. The system shall prevent unauthorized access to user accounts. [C]
13. The system shall prevent unauthorized viewing of password files. [C]

14. The system shall prevent unauthorized modification of password files. [I]
15. The system shall detect changes made to users' password. [I]
16. The system shall prevent unauthorized deletion of password files. [I]
17. The system shall prevent unauthorized viewing of the event logs of changes made to files. [C]
18. The system shall prevent unauthorized modifications of the event logs of changes made to files. [I]
19. The system shall prevent the unauthorized viewing of the event logs of users. [C]
20. The system shall prevent the unauthorized modification of the event logs of users. [C]

## 3.3 Essential Security Elements

*Authentication*:
Authentication allows us to identify who is allowed to access our system. It is the first barrier in preventing the threats listed above from accessing our system and compromising the confidentiality and integrity of our file sharing system. Only legitimate users of the system are allowed to access the system and we would like to keep any threats from accessing the system in the first place. Authentication is therefore essential in providing a first line of defense against any intruders.

Furthermore, authentication also allows us to bind principals of our system to the actions that they perform. Each user of the system is associated with a user ID, and whenever an action is performed by a user, this can be recorded in an event log. Any malicious actions by insiders or a hacked account can be traced with the event log. Thus, authentication can hold users accountable for their actions.

*Authorization*:
Authorization is another critical aspect of our system because there are various types of users in our system. We want to differentiate between the different kinds of actions that different users can perform, so a mechanism is needed to determine what actions can be performed. A user can either intentionally or unintentionally cause harm to the system if given too many permissions. In our system, authorization is crucial because we do not want users to have access to assets that they should not view or modify as we must uphold the principle of least privilege. Appropriate authorization must also be given to legitimate users to ensure the availability of files to these users.

*Audit*:
As mentioned in authentication, we require audit mechanisms such as event logs that can record and review actions. This allows us to hold users accountable for the actions that they perform in the system. Malicious users who abuse the permissions given to them will have their

account suspended. This audit mechanism is important in deterring potential attackers as well as ensuring that users of the system act responsibly. Furthermore, apart from deterrence, monitoring the event logs can allow the admin to spot any suspicious actions that have been performed. This allows the admin to take the appropriate measures such as performing a system rollback in order to revert the system back to a safe state. These reasons demonstrate the importance of the audit mechanism in our system.

*Confidentiality*:
Confidentiality is key in our system as well. Since our system implements file sharing, users who upload personal files would naturally want to keep their files secret and secure. This makes confidentiality important because we would want to protect these files (assets) from any unauthorized disclosure. A key feature of our system is that a user is able to share files to a certain group of users. However, at the same time, the user who shares the files might not want other unauthorized viewers to see the file. For example, a team that has their code stored using this system will not want other non-team members to see their code. Given that some users want to share sensitive information with a restricted group of people, confidentiality is therefore an important security element that we have to implement for our system.

*Integrity*:
Integrity is also integral to our system. Users who do not have the right to edit files should not be able to make changes to files - owners should be assured that only editors have the right to make changes to files. Aside from privilege restrictions, our system also detects all changes and records it in an event log. This will provide some deterrence through accountability.