

Securing the Cloud - Report



Introduction

The task of this assignment was to develop a secure cloud storage application for Dropbox, Google Drive, Office365 etc. The application must secure all files that are uploaded to the cloud such that only people who are part of your “Secure Cloud Storage Group” will be able to decrypt your uploaded files. To all other users the files will be encrypted. The application should implement a suitable key management system that will allow files to be shared securely and allow for users to be added and removed dynamically.

- Brandon Dooley (#16327446)
-

Approach

I decided to build my application using ReactJS to design my front end, I chose this as it is a perfect framework for developing single-page dynamic web applications and I had quite some experience working with it. For my file management system I used Google's Firebase platform which allowed for me to both store files and manage my user and key management system via a NoSQL database.

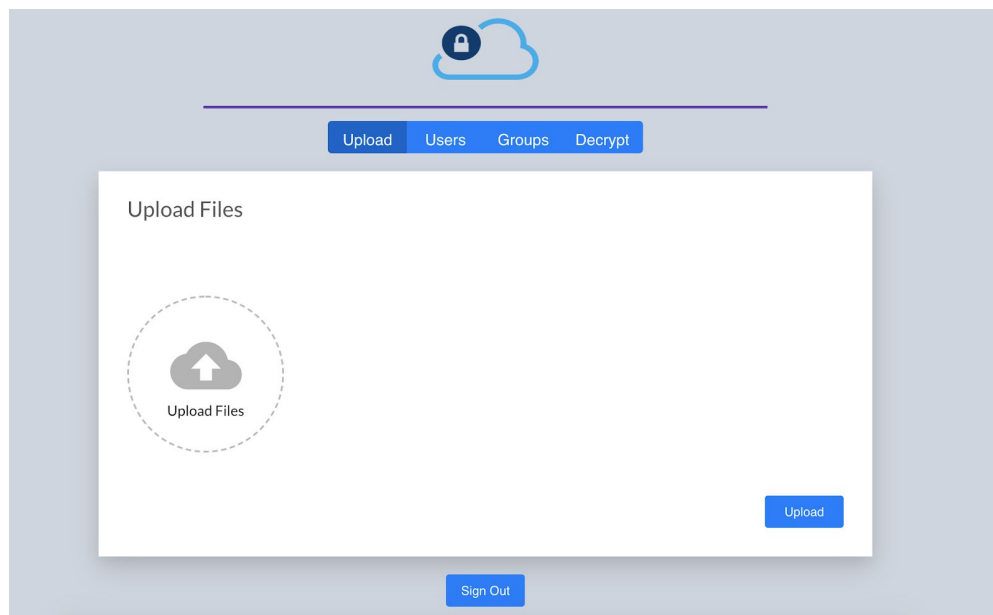
I also availed of public libraries during my development such as:

- Bootstrap
- CryptoJS
- Semantic-UI

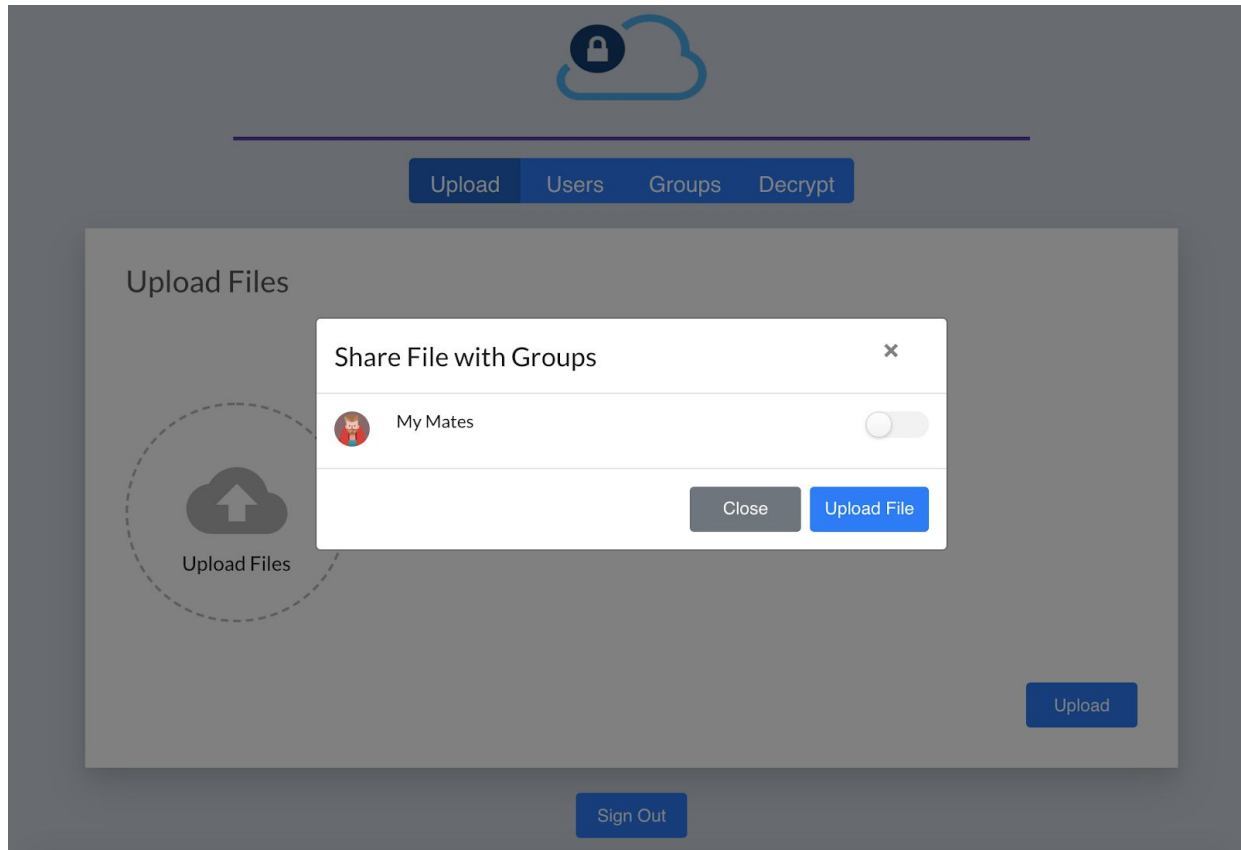
Design

Uploading Files

When a user signs into the application they are initially presented with a screen in which they can upload files using the secure cloud storage application. They also have the option to navigate to multiple other screens such as *Users*, *Groups*, and *Decrypt*.



Once a user has selected the files they wish to upload they are then prompted with a modal that enables them to share the files they are about to encrypt with a group created on the application via the *Groups* screen.

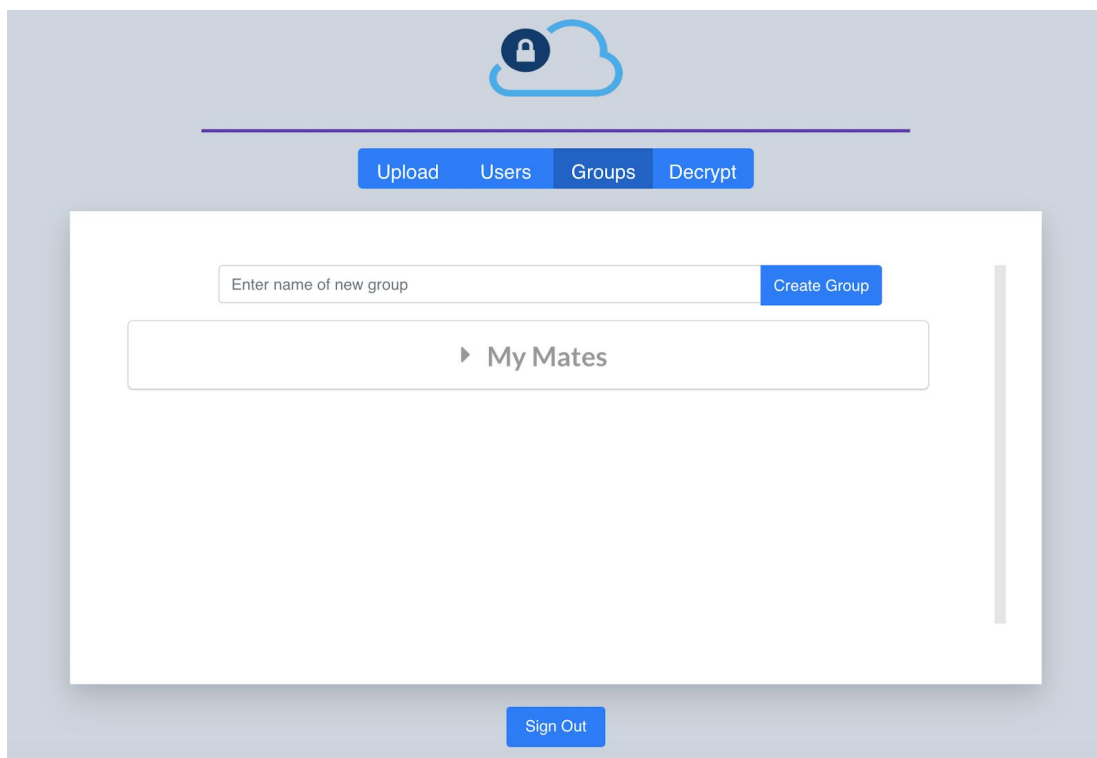


The file is then encrypted on the client side using the open source library *CryptoJS*, further details on the encryption and key management methods will follow. Once the encryption is completed and the file has been uploaded to Firestore a success progress bar is displayed to the user. They can then navigate to the *Groups* screen to view the encrypted file or decrypt the file.

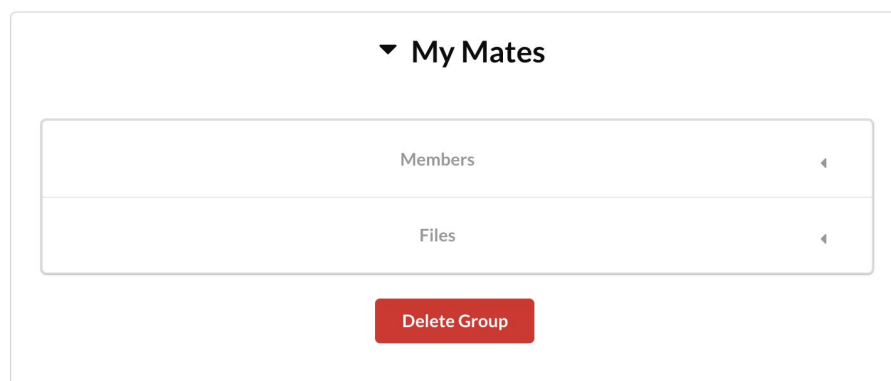


Viewing Files & Groups

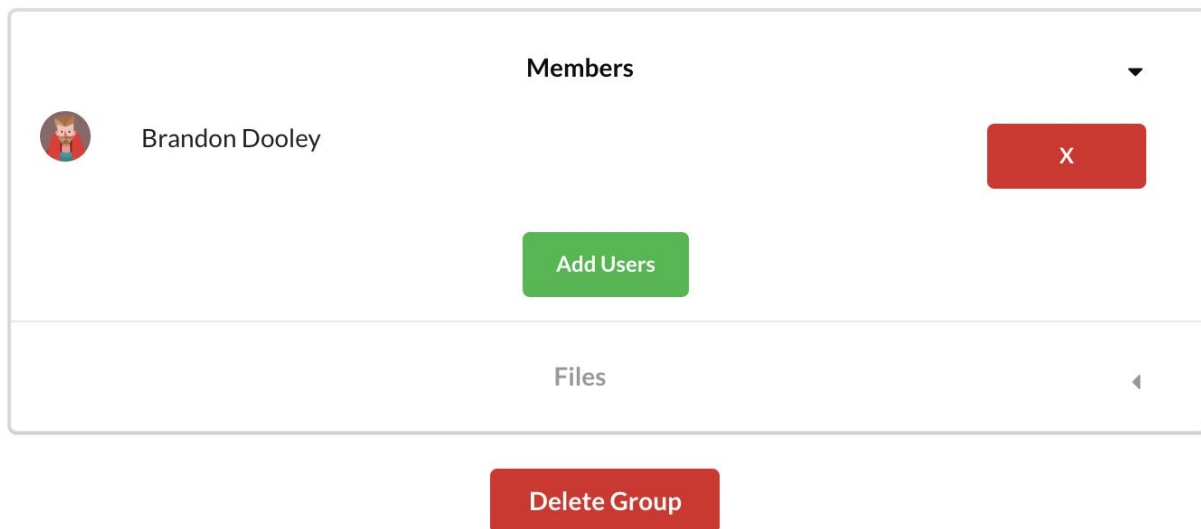
When a user navigates to the *Groups* screen they are initially presented with all groups that they themselves are members in. They also have the option to create a new group on the application.



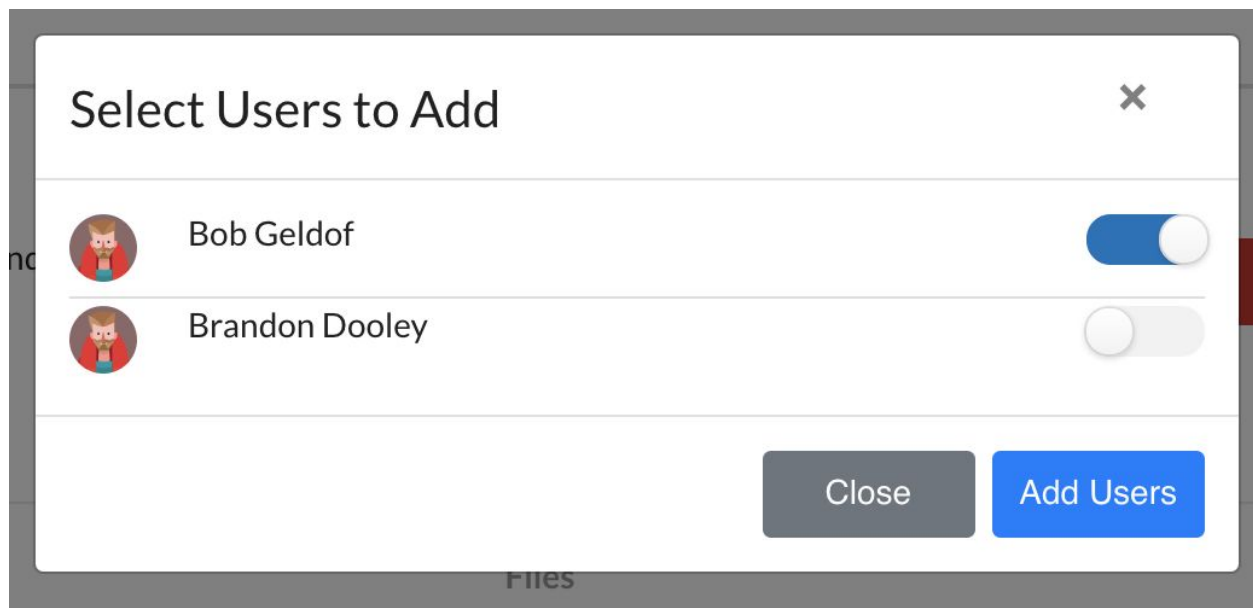
From this screen the user can expand the group dropdown menu to explore group members and files.

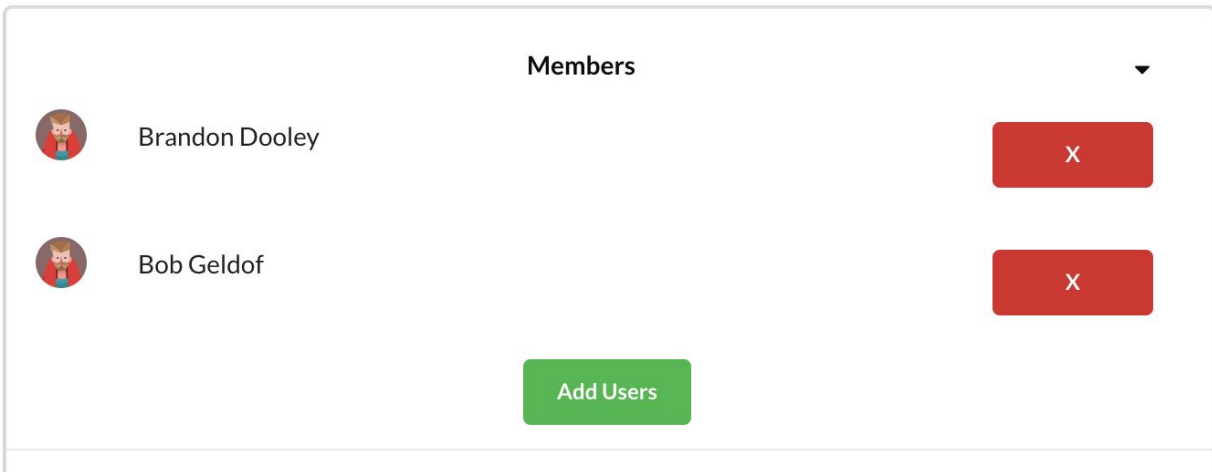


Within the members dropdown the user can view all current members within the group.

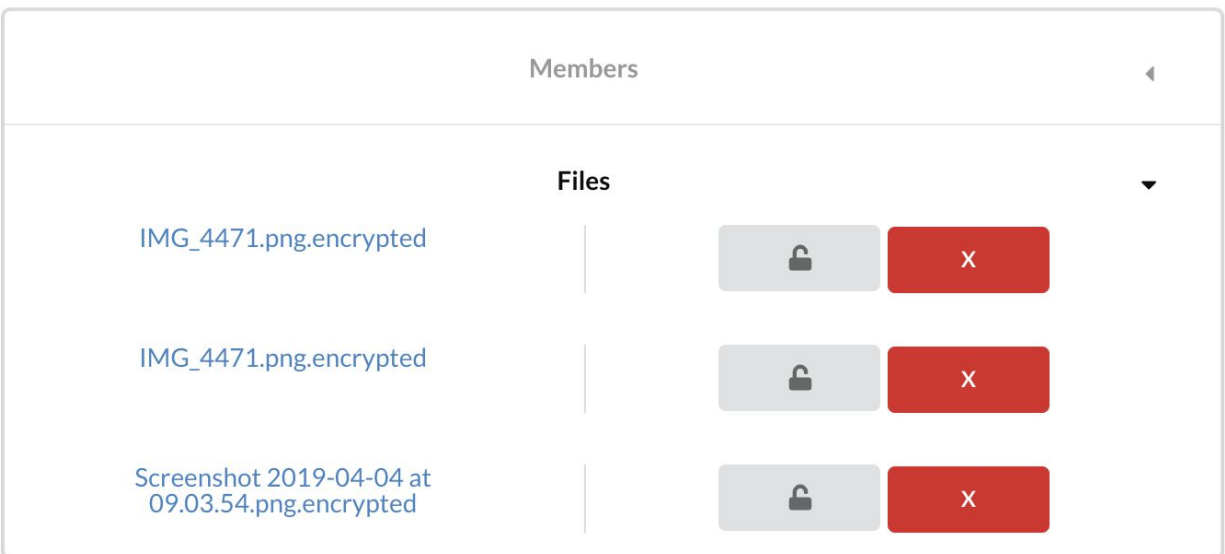


Through this dropdown they can also add new users to the group, which will enable them to see all files shared with the group. The user can also remove users from the group, by doing this the removed user will no longer be able to access this group nor the files associated with it.





Within the files dropdown the user can discover all encrypted files associated with the given group.



From this menu, the user then has three options. They can download the encrypted file by clicking on the blue link with the file name, they can decrypt the file by clicking on the grey unlock icon or they can delete the file by clicking on the red X. Once a file has been deleted.

Encryption & Key Management

All encryption is performed on the client (browser) side. When a user has selected the files they want to upload and also the groups they want to share the files with the client begins encrypting the files and then uploading them to Firestore.

The file is first read into the client using a [FileReader](#). This allows for the application to process the file as a [Data URL](#) string which can be easily encrypted, uploaded and decrypted. The actual decryption itself is performed using a Javascript library called [CryptoJS](#), this is a library which contains many of the standard cryptography algorithms needed in order to encrypt a file.

I chose to encrypt the files using [AES](#), a symmetric key encryption algorithm. The implementation of the algorithm was provided by the CryptoJS library. A random key, which is associated with each file is created by generating a [SHA-1](#) hash of a combination of the current date string with a random string. The file is then encrypted using AES with this random key as it's symmetric key.

```
async encrypt(file){
  return new Promise((resolve, reject) => {
    const reader = new FileReader();

    reader.onload = (e) => {

      // Extract file data from FileReader
      var file = e.target.result;

      // Generate random hash using date and random string
      var current_date = (new Date()).valueOf().toString();
      var random = Math.random().toString();
      var key = Crypto.createHash('sha1').update(current_date + random).digest('hex');

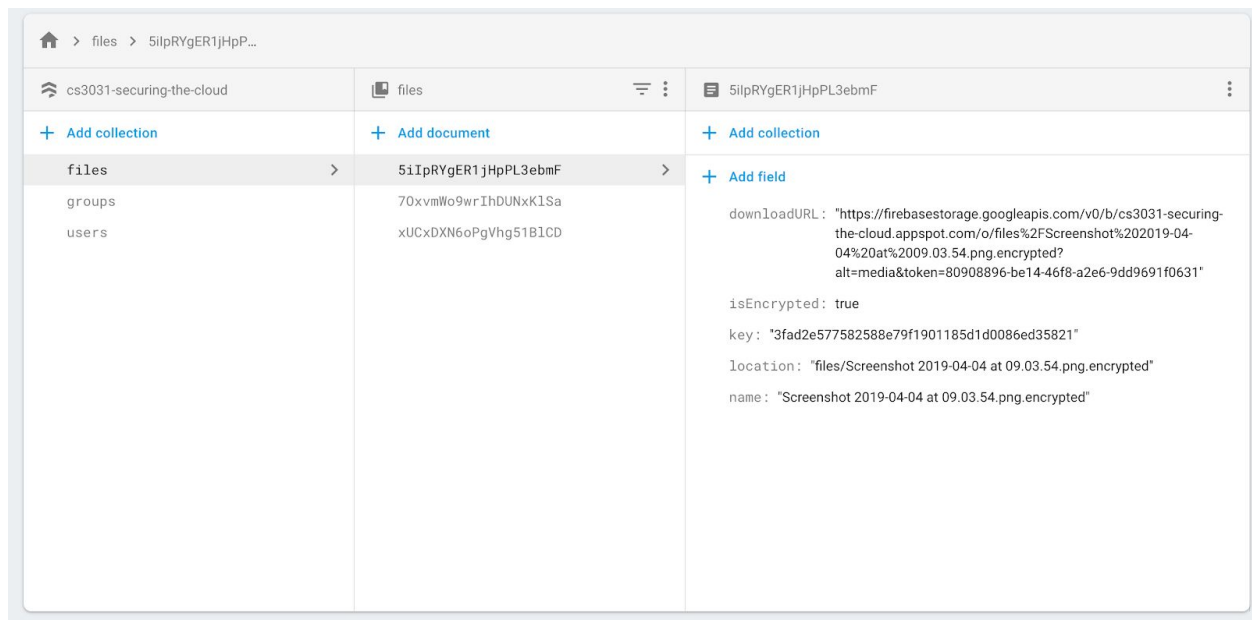
      // Encrypt file data using key
      var encrypted = CryptoJS.AES.encrypt(file, key);

      // Return file-key pair
      var fileKeyPair = {
        file: 'data:application/octet-stream,' + encrypted,
        key: key
      }

      resolve(fileKeyPair);
    };

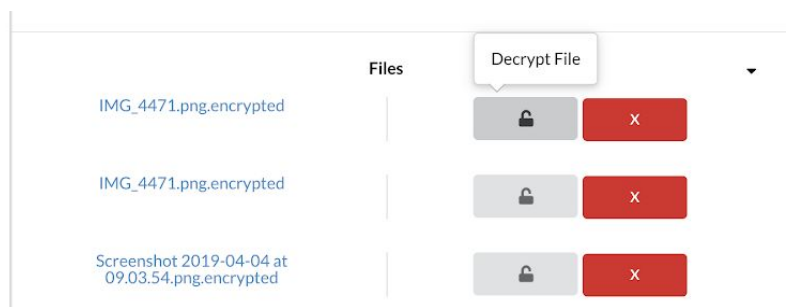
    reader.readAsDataURL(file);
  });
}
```

Once each file has been encrypted they are then ready to be uploaded to Firestore and shared with each group. Each file is uploaded to the Firestore storage application. A reference to each file is also created within the Firebase database. This is done in order to serve as a key management system. Each file's metadata and its associated key is stored within an index in this database. This information is never exposed to the end user and is only accessed by the client when decrypting the file. Once a file has been deleted it is removed from both the Firestore and Firebase providers.



Decryption

When a user wants to decrypt a file they must first navigate to the *Groups* screen and then to the *Files* dropdown for the group that the file has been shared with. From here the user can then click on the decrypt icon which will trigger the decryption flow.



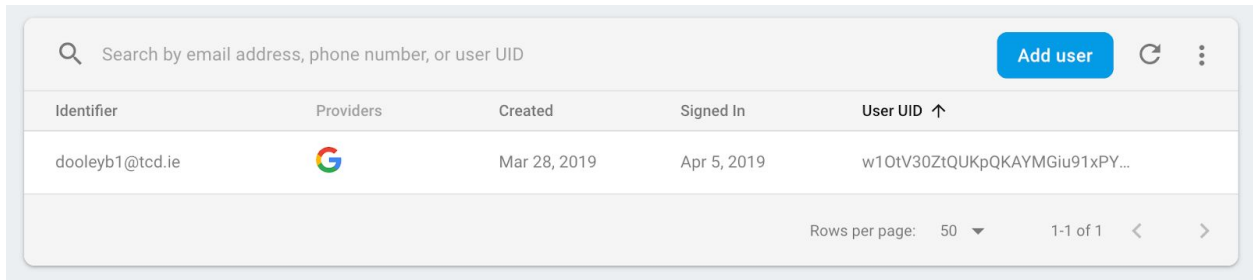
Since AES is a symmetric key encryption algorithm, once we know the key associated with the encrypted data it is quite straightforward to decrypt it. Once a user has selected a file that they would like to decrypt the client first fetches the key associated with the file from the relevant Firebase index. The client then downloads the encrypted version of the file and similar to how it was encrypted uses a FileReader to load the file as text.


```
async decryptFile(file){  
  
    // Create FileReader and XMLHttpRequest objects  
    var xhr = new XMLHttpRequest();  
    var reader = new FileReader();  
  
    // Download encrypted file  
    xhr.responseType = 'blob';  
    xhr.open('GET', file.downloadURL);  
    xhr.send();  
  
    // XHR Callback  
    xhr.onload = function(event) {  
        reader.readAsText(xhr.response);  
    };  
  
    // Reader callback  
    reader.onload = function (e) {  
  
        // Decrypt file using key  
        var decrypted = CryptoJS.AES.decrypt(e.target.result, file.key).toString(CryptoJS.enc.Latin1);  
  
        // Ensure file is not corrupt  
        if(!/^data:/.test(decrypted)){  
            alert("Invalid pass phrase or file! Please try again.");  
            return false;  
        } else {  
            console.log("Successful decryption", decrypted);  
        }  
  
        // Handle decrypted file download here  
        fileDownload(decrypted, file.name.replace('.encrypted', ''));  
    }  
}
```

Then, using CryptoJS it decrypts the ciphertext using AES and the file key. It then converts the decrypted text back into a file object based on the decrypted DataURL and allows the user to then proceed with downloading the decrypted file.

User Management

To manage my applications users I took the approach of using Google's Firebase platform. This gave me the ability to allow users to onboard to my application using a Google account which was also the platform used by the college.



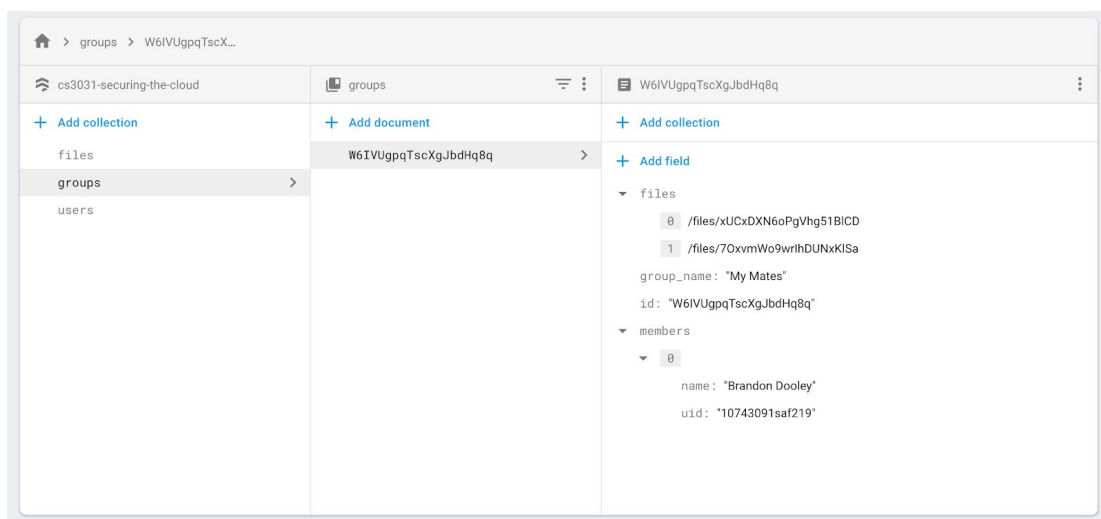
Identifier	Providers	Created	Signed In	User UID ↑
dooleyb1@tcd.ie		Mar 28, 2019	Apr 5, 2019	w10tV30ZtQUKpQKAYMGiu91xPY...

Rows per page: 50 1-1 of 1

Firebase also allowed me to store and manage data regarding users, such as what groups they belonged to etc in a NoSQL database. This structure allowed for efficient queries which enabled me to develop a real-time responsive web application that met all of the desired functional requirements.

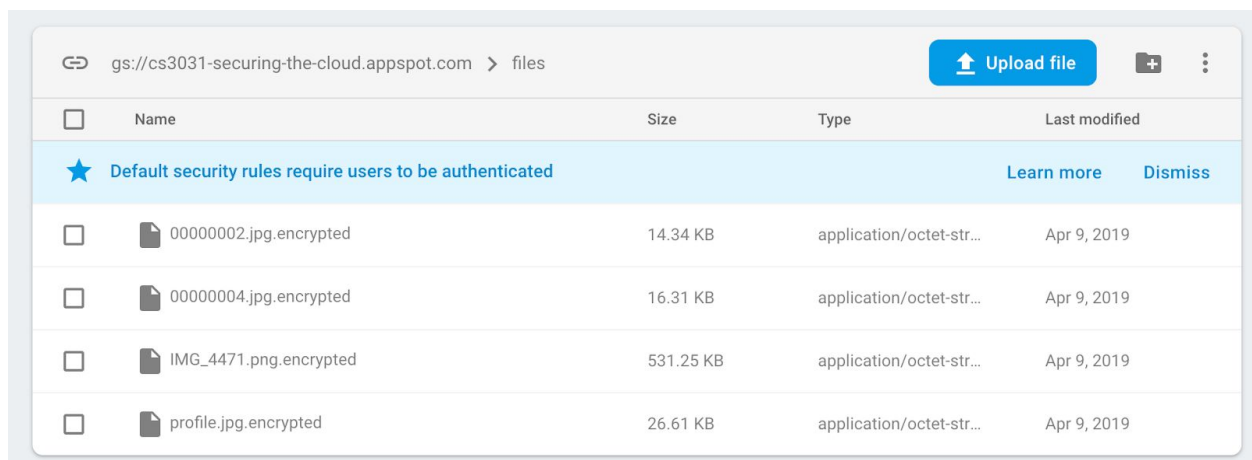
Group Management

Similar to users, when a group was created within the application an entry was created within the Firebase database containing relevant information for the group such as users, files, group name etc.

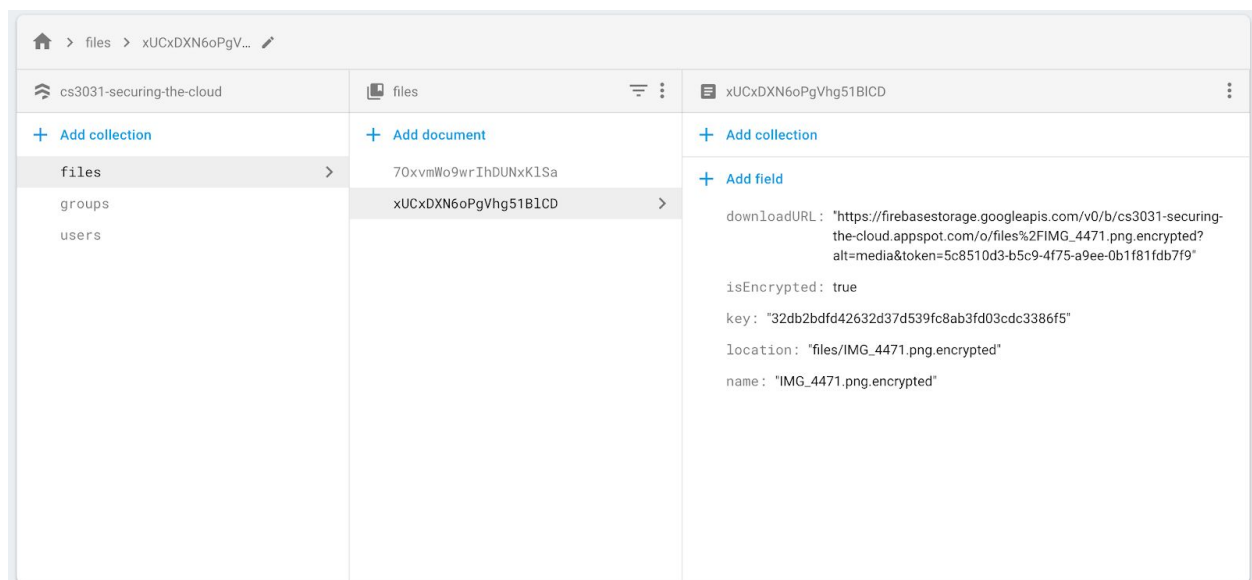


File Management

When a user had uploaded a file via the web application, the encrypted file data was uploaded to Google Firestore platform which allows for files to be uploaded and downloaded easily.



In order to manage which files belonged to which group an entry was also created within the Google Firebase database which allowed for files to be easily shared with multiple groups whilst persisting one given location. This removed the need for files to be uploaded multiple times and also allowed for the removal of files across multiple groups.



The code for this project is a few thousand lines long across several different files and sub-directories. The important encryption and decryption parts have been included. The rest of the source code can be found here:

GitHub Repository: [cs3031-cloud-security](#)

- Brandon Dooley (#16327446)