**Abstract**

Security and privacy are hot topics now. Today there are many systems that store our files but you are definitely not sure whether your files are 100% safe. The reason is you are storing the files directly into their servers, which is clearly not the one you want. On thinking about the way we can store data securely, we found a model that encrypts the file in client side without even referring the server. Therefore, the file that you will get is simply an encrypted file that is going to be stored in client machine. The application that we built is a responsive web and a hybrid mobile application, which means that it will work the same in all screens and in all platforms, whether it is IOS, Android or Windows, the application is same. We found issues with the system that we built but we also found the solution for the same. When it comes to encrypting data and securing information, people naturally expect the page to be loaded through HTTPS. In this case, I believe it is not necessary, as apart from the initial download of the HTML and assets, no data is been transferred between you and the server – everything is done client-side with JavaScript. User who encrypt the file does not need to remember the password all the time. They can provide some questions which gives the password as the answer. No worries we are not encrypting the file simply from the answer for those questions. We are generating a random number for each user and doing a simple calculation based on the random number. Thus, every time when the user does an encryption operation a random number is been added with the passphrase and the encrypted file will be available for them. Thus even if they remember the answer for those questions, they cannot see your file unless they login. We can share the encrypted files only with few group of people thus providing access only to them and creating a secured workgroup. Thus, we do not need to worry about the file that they are storing in different servers because they anyway need to use the above model to decrypt it thereby creating the most secured file system.

**Firebase Authentication:**

Most apps need to know the identity of a user. Knowing a user's identity allows an app to securely save user data in the cloud and provide the same personalized experience across all of the user's devices.

Firebase Authentication provides backend services, easy-to-use SDKs, and ready-made UI libraries to authenticate users to your app. It supports authentication using passwords, popular federated identity providers like Google, Facebook and Twitter, and more.

Firebase Authentication integrates tightly with other Firebase services, and it leverages industry standards like OAuth 2.0 and OpenID Connect, so it can be easily integrated with your custom backend.

**Firebase Realtime Database:**

The Firebase Realtime Database is a cloud-hosted database. Data is stored as JSON and synchronized in realtime to every connected client. When you build cross-platform apps with our iOS, Android, and JavaScript SDKs, all of your clients share one Realtime Database instance and automatically receive updates with the newest data.

**Key Capabilities:**

1. **RealTime:** Instead of typical HTTP requests, the Firebase Realtime Database uses data synchronization—every time data changes, any connected device receives that update within milliseconds. Provide collaborative and immersive experiences without thinking about networking code.
2. **Offline:** Firebase apps remain responsive even when offline because the Firebase Realtime Database SDK persists your data to disk. Once connectivity is reestablished, the client device receives any changes it missed, synchronizing it with the current server state.
3. **Accessible from Client Devices:** The Firebase Realtime Database can be accessed directly from a mobile device or web browser; there’s no need for an application server. Security and data validation are available through the Firebase Realtime Database Security Rules, expression-based rules that are executed when data is read or written.

**How it works?**

The Firebase Realtime Database lets you build rich, collaborative applications by allowing secure access to the database directly from client-side code. Data is persisted locally, and even while offline, realtime events continue to fire, giving the end user a responsive experience. When the device regains connection, the Realtime Database synchronizes the local data changes with the remote updates that occurred while the client was offline, merging any conflicts automatically.

The Realtime Database provides a flexible, expression-based rules language, called Firebase Realtime Database Security Rules, to define how your data should be structured and when data can be read from or written to. When integrated with Firebase Authentication, developers can define who has access to what data, and how they can access it.

The Realtime Database is a NoSQL database and as such has different optimizations and functionality compared to a relational database. The Realtime Database API is designed to only allow operations that can be executed quickly. This enables you to build a great realtime experience that can serve millions of users without compromising on responsiveness. Because of this, it is important to think about how users need to access your data and then [structure it accordingly](https://firebase.google.com/docs/database/web/structure-data).

**Existing System:**

Until today many digital storage are available in the world. Everything we store there considered being secure. The maximum that can be done is just upload the data and send it to the other person for sharing purpose but we does not know who all will see the data. Thus, we can say the file is compromised. The maximum we can do is to password protect those files but sadly those passwords will also been stored in their database. The main problem here is there is no single application that does not store your password in their database. This tells there is always a vulnerability in the way they store if the password is compromised.

**Disadvantage:**

1. The file is not secure as the password is stored in their database
2. Currently they are building three applications in a minimum like one for website, one for mobile websites and one for mobile application (may be more based on the number of OS they are supporting).
3. If files are password protected in your desktop then there is also a problem where human can remember a maximum of 5 to 6 passwords which is again a huge problem.

**Proposed System:**

The main aim of our proposed system is not to store the password in database. We also taken into consideration that human can easily remember words that are related to incidents. For example: 24 is my lucky number, If I store password as 24 then there is a chance that I may forgot it. But if I put a security question like “My Favourite number” on top of it then he/she can easily get the password. Thus making it more simple to remember. Thus for every encrypted file there will be a filename and a security question. Thus even if the person forgot the passphrase he can easily decrypt it by flashing the security question to him. If the passphrase is leaked and the encrypted file is also with a wrong person, even then that person cannot decrypt the file. As we are creating a random number with the file and that is stored in our database for that particular person, only, when that person knows the victims’ gmail credentials that person can login to the system. Getting to know the gmail credentials of a particular person is highly impossible.

**Advantages:**

1. The file is highly secured as the passphrase is not stored anywhere
2. Single code base for website, mobile website and mobile application. Thus making the life of developer easier.
3. The recovery is much easier with a simple security question feature.