Password Keeper White Paper

On the frontend, when a user signs up, the user must first pass some checks for the password, so that it is at least 8 characters and not the same as the username or email. Once this passes, a salt is generated which is 32 bytes and then a hash is generated from the salt concatenated with the plaintext password. A salt is randomized data so that a password which is used by two users will not produce the same hash, so that if the database is leaked, rainbow table brute-force attacks are not useful. Whenever the user goes to login, the backend returns a challenge and the salt. From this, the login function in php is called with the password, formed from a hash of the salt and the plaintext password, and the challenge is sent back. The backend will then verify that the challenge is identical, and that the salted and hashed password matches, and if this is the case, the user is authenticated. On the frontend, the hash is taken of the plaintext password, and this will be used for encryption. Ideally, there would be a second salt for this hash used for encryption, but there is no possibility to add this since the database schema cannot be modified. This hashed password will be saved in local storage.

Before every backend function is called, a preflight function is called to verify that the request is authenticated. To reduce the likelihood of DDoS attacks, the origin field of the header is inspected in the preflight. According to RFC 6454, if the origin is not a scheme/host/port triple, then the origin field is set to null. It could have originated from a file on a client device, which may be a sign of malicious intent such as DDoS attack. Thus, if origin is null, we return 403 forbidden. In the preflight, a web session token is used, and a user session cookie is used. If the web session is not present, then the preflight will create a token if it is a login or signup operation which is occurring, otherwise, the user will be told to authenticate again. If the web session actually was present, then it would be checked to see if it is expired. If it is, the user will have to authenticate again. If it is not a login or signup operation, then the user session will be checked, and if it is expired, the user will have to authenticate again. The web session lasts for 12 hours and the user session for 15 minutes. This means that the user will have to authenticate again if they are inactive for 15 minutes or more, and will have to authenticate again after 12 hours whether or not they were active the entire time or not. There is another session type: the login session. This session lasts for 30 seconds, and is created when the user attempts to login, with the challenge and salt being sent back to the front end. The login session ends when the user logs in, or once 30 seconds has passed, whichever is sooner.

The site passwords which are created by the user are encrypted using AES CBC encryption, which takes a 16-byte initialization vector (IV) and the hashed password, which is not salted because the salted version is stored. Whenever the user requests to access the password, it is then decrypted using the same hashed password and IV. The same is done with the site username, which is also encrypted.