Epic UFC-257: Client Side Encryption

The following document will describe a design that could be used to implement client side encryption (aka e2e encryption) in uFincs.

Acceptance Criteria

- I want all of my personal data (accounts, transactions, import profiles, etc) to be encrypted using my 'password', so that the server (aka uFincs, aka the operators of uFincs) never have access to my raw finances.
- From my perspective as the user, very little should change from a UI/UX perspective.
- I want to know that I can't reset my password without losing all my data.
- I want to be able to change my password while still maintaining my data.
- I want a loading screen when first loading the app to show that my data is being fetched and decrypted.
- I don't want a loading screen after my data has been cached client-side; it should just fetch and decrypt from the backend in the background.
- The encryption/decryption processes should work in a background thread so that the UI thread isn't too heavily impacted. It should also, ideally, be multi-threaded for even faster performance.
- The encryption process needs to be more-or-less a self contained library, so that we can open source it.
- Along the lines of a self contained library, the encryption process should require minimal (ideally, none) changes to the data schema. I should not have to rewrite my schema to use blob fields instead of regular data fields.

Note: On the point of not having to change the data schema, this is because this epic can be thought of as the first of potentially multiple related to encryption. Since this is the first, we're aiming for the lightest weight implementation of e2e encryption that we can come up (i.e. the MVP). We want to just wrap our existing architecture/schema with encryption capabilities, not twist them to accommodate encryption.

This mostly means using field level encryption instead of something more akin to row level encryption. While field level encryption is bound to be less performant, it is also more flexible, as well as simpler to implement on top of our existing architecture.

Research

A large amount of research has already gone into how we might go about implementing the client side encryption process. It can be found at Idea: Client Side Encryption.

Pieces of the Solution - Phase 1

Having already thought a *lot* about the problem, I think I know more-or-less what things we will need to create the MVP in order to solve this problem. They are as follows:

- An open source-able library that consists of two main parts:
    1. A wrapper over the WebCrypto API that handles the actual encryption/decryption/password hashing components.
    2. A Redux middleware that integrates said wrapper into our existing Redux infrastructure.
        - This would have a sub-part that leverages Web Workers for performing the encryptions/decryptions.
- A new column in the `users` table for storing the encrypted DEK (data encryption key).
- A new column in the `users` table for storing the salt of the KEK (key encryption key).
- Modifications to the sign up process to generate the DEK.
- Modifications to the Backend users/authentication service to include the encrypted DEK when logging in.
- Modifications to `createRequestSlice`/`createOfflineRequestSlice` that allow adding meta tags to the request actions.
- Some way to migrate (my) data from the old database to the new encrypted format.
- A loading screen that plays during the initial app boot process while fetch/decrypt the user's data.
- A form for changing my password (in the user settings).
- A warning when signing up that forgetting your password means losing all your data.
    - I think this could be implemented in the form of a checkbox on the Sign Up form with something along the lines of "I acknowledge that losing/forgetting my password means that all of my data will be lost".
- A "Forgot my password" feature that allows a user to change their password but also wipes their data (and let's them know about this).