DataStructures:(encryption and integrity of corresponding structs explained below):
For anything that has a mac we will always check the integrity of the data before using or appending data.

- Access tree:
    a. Each node is a struct which contains:
        i. Username of who has access
        ii. Usernames public key encryption of symmetric key used for encryption of the file data header
        iii. Usernames public key encryption of mac key for data header
        iv. The username of the sharer/signer(only root can revoke access).
        v. Signed encrypted value of encrypted symmetric key and mac key, and username signed by the digital signature of sharer/signer above.
- File header metadata:
        i. Symmetric key used to decrypt the file
            1. Is originally generated by the owner using the random byte generator function
        ii. Mac key used to hmac the file once done editing
            1. Also generated using random byte generator function
        iii. UID of actual file location
        iv. Current position in file
        v. Owner of file
        vi. Mac of encrypted header data(in order to get this we enter all of header data excluding the macs into hmac function).
        vii. Mac of actual file itself.
- User struct:
        i. Username
        ii. Dictionary(or map depending on golang structures)where the key is a hash of encryption of the filename which maps to the uid of the access tree, as well as a pointer to the metadata header for the file.
        iii. Their secret key that corresponds to their public key encryption
        iv. Digital signature key
        v. Mac of the user struct

1. How is a file stored on the server?
    a. When the file is first created we are going to also create a meta file and access structure to go along with it. (add yourself to the owner field, as well as to the access structure as the root node.)
    b. We will then generate a symmetric key and mac key using the random byte generator functions and place these keys inside the meta file data header. These keys are used for encryption/decryption and verifying the actual file contents itself.
    c. The symmetric keys for encryption and the mac of the file_header will be stored in the access token tree structure at the corresponding user node. Both also generated by the random byte generator function.

    d. The actual mac of the file data contents and file header data contents will be stored in the meta data header. For both we will pass in the encrypted contents of the file data or header into the HMAC cryptographic function.
    e. On creation we also generate a uid for this file itself using the filename and a randomly generated salt. This salt will We will also generate a uid for the file header using filename concatenated with the phrase header and a randomly generated salt. Random byte generators seems like a great salt.
2. How does a file get shared with another user?
    a. The user who is sharing first generates an access token for the receiving user which is the UID to the access token tree structure and sends it to the receiver. They also send the UID to the Meta file data header. The sharing user also adds them to the access token tree structure, making sure to correctly encrypt everything with the receiving users public key. Defining the fields as mentioned in the data structure section. The user adds the as hash of the encrypted file name to their filename dictionary, and corresponding to this is an encryption of the UID of the access tree structure as well as an encryption of the UID of the file header meta data that they received.
3. What is the process of revoking a user's access to a file?
    a. In order to revoke access we make sure that the current user is the owner of the file. If they are the owner then we find the requested user in the access tree and remove that user. We then generate a new symmetric to put into the file header structure, as well as a new mac key. And re encrypt the file and its contents using this key. And then re mac the encrypted file contents using the mac key. And place them in the meta file structure. We also then generate a new symmetric key to decrypt the meta file and a new mac key for the meta file , and re encrypt this symmetric key and mac key in the access token structure for every valid user using their public key, and also sign the encrypted key using your own secret key as well as adding a message that states who signed the message(this will always be root in this instance).
4. How does your design support efficient file append?
    a. It supports efficient file append because we store a file fd position. So we can immediately jump and append the data to the existing file. We only need to encrypt this additional data and append it to the existing file. Then mac then re hmac the whole file using the shared hmac key.
5. How does a user encrypts its data?:
    a. User will encrypt the data stored in its user struct using the password based key derivation function however to ensure no two users have the same key we add a salt deterministically generated based on their username to their password before encryption. It will also sign its own data using its digital secret key and check with its own public key if the contents of the user struct has been edited. Looking at the mac. By inputting all the values of the user struct into the verify function.
6. User's public key and digital signature public keys:
    a. Will be stored in the Keystore at keys(indexs) <username>enc for the public key for encryption and <username>sig for digital signatures. If it was username was bob keys would be bobenc and bobsig.

Security analysis:

1. Attack one :A malicious user gets to share the file but later its access gets revoked. However they stored the symmetric key and mac key for the file, as well as the symmetric and mac key for the file header! And they also stored the uid of the access structure. The revoked user then shares the file with another user. The receiver tries to decrypt the file but it doesn't work. Then they try to decrypt the file_header it also doesn't work, because the file_header was re-encrypted with a new key, the sharer didn't have access to. Finally they use their access token to get to the access structure and try to decrypt their node. But their node is not valid because its not there!The receiver can not get access to the file or read any of the contents.
2. Attack three:
   To prevent an attacker that has a password bank of common passwords from guessing a password that allows them to access the user struck we add a salt to the password. This salt is deterministically generated based on the username of the user trying to get into the user struct. This ensures that access into the user struct cannot be easily guessed by an attacker with knowledge about passwords of the current users
3. Attack four:
   A file is shared with a malicious user. The malicious user then learns the location of the access tree. They then get their access revoked. Afterwards in an attempt to mess with the user they change the value of certain users access nodes, thereby changing the value of their keys for encryption and the mac. However, when the user checks the access tree to see if their keys have been updated they check that the user who modified the keys is actually in the access tree as well. If so they then use that users public key signature to verify that it is actually from that user. Therefore this attack will be detected.
4. Attack four:Attack five: A malicious user has access to the file and validly changes the mac and encryption key for the file header but then does not update the other users. And re encrypts the file and the file_header. Now when the other users try to edit the file they will first check for new keys in the access token structure and see that none have been given. Next they will check the integrity of the file header in order to get access to the file but since the mac will no longer match they don't do anything and will no longer append data. However consider that the malicious user does update all the users keys and is now in control of the access keys. This does nothing as they are in the exact same place as they were before.
5. Attack two: When a file is shared the sharer sends an access token to the receiver containing the receiver's public key encrypted position of the access tree to the receiver. If the attacker overwrites the access token it will point to a different location and won't leak any information used to gain access to any other information. The user will just not be able to retrieve any keys.