Things to remember:

- ***UUID doesn't hide the input bytestring to create the UUID*** - That is to say, you should not pass a confidential secret key as the byte slice here and use the resulting UUID publicly; an attacker may figure out information about the secret key from that UUID.
- ***Datastore is untrusted*** - guarantee the confidentiality and integrity of any sensitive data or metadata you store on it.
- ***Keystore is trusted public key server*** - allows us to post and get public keys, You can assume no aacker will overwrite any entry you add to the Keystore
- ***Implement authenticated encryption (strongly recommended)*** - It will greatly simplify the design if you implement an authenticated encryption scheme here, which will be using HMAC, HKDF, and symmetric encryption in an interesting way.

Part 1 Requirements

- **Authentication/confidentiality(?)** - Any data placed on the servers should be available only to the owner and people that the owner explicitly shared the file with, not the server.
    - The server and other users should not learn the file content, the filename, and who owns the file, unless one shares the file with them.
- **No need to hide file length** - The server is allowed to know the length of the file content you store, which you don't need to hide
- **Check integrity** - If the server modified the data you stored, you should be able to detect that and trigger an error. It is okay if the server returns a previous version of the file, which the client does not need to detect

1. How is each client initialized?

- Initialize a new User struct with Username = username.
- Generate signing key and verifying key for digital signature using DSKeyGen() (DSSignKey, DSVerifyKey, error).
- Generate decryption key and encryption key for asymmetric encryption using • PKEKeyGen() (PKEEncKey, PKEDecKey, error).
- Store public keys in the Keystore using KeystoreSet(key string, value PKEEncKey/DSVerifyKey) error.
- Store private keys (DSSignKey,PKEDecKey) in the User struct.
- Store User struct in the Datastore using DatastoreSet(key UUID, value []byte), possibly using StoreFile(filename string, dat []byte). Need to guarantee the confidentiality and integrity of the information. So we need to generate a UUID from username/pw that is hard to guess.
- GetUser returns the pointer to User
- To make sure user entered right password, a way to verify password is to store {username, Salt, H(P||Salt)}.

2. How is a file stored on the server?

- StoreFile places the file in DataStore (possibly in User struct?)
- Update the User struct back to Datastore after making changes
- When returning data, check if file exists and is untampered (integrity) - so check hash
- Make sure to return latest version of file
- AppendFile should add to file without redownloading/decrypting - need clever solution to store data (like CBC? I dunno)

3. How are file names on the server determined?
    - Based on both username/UUID and file name?

4. What is the process of sharing a file with a user?

5. What is the process of revoking a user's access to a file?

6. How were each of these components tested?

InitUser()

- Initialize a new User struct with Username = username.
- Initialize a map, FileToUUID, in the User struct whose (k,v) = (key(filename, username, password), UUID_FILE)
- Generate signing key and verifying key for digital signature using DSKeyGen() (DSSignKey, DSVerifyKey, error).
- Generate decryption key and encryption key for asymmetric encryption using • PKEKeyGen() (PKEEncKey, PKEDecKey, error).
- Store public keys in the Keystore using KeystoreSet(key string, value PKEEncKey/DSVerifyKey) error.
- Store private keys (DSSignKey,PKEDecKey) in the User struct.
- Store username and password in the User struct
- Store User struct in the Datastore using DatastoreSet(key UUID, value []byte), possibly using StoreFile(filename string, dat []byte). Need to guarantee the confidentiality and integrity of the information. So we need to generate a UUID from username/pw that is hard to guess.
- Convert the new User struct into a byte[] using json.Marshall(_).
- Generate a base key using Argon2Key(password, username, 16)
- Generate UUID using HMACEval(Base Key, “randomstring1”)
- Create EncUserData, the encrypted User struct and tag: EncUserData = AuthEnc(Base Key, User)
- Generate SymmKey using HMACEval(Base Key, “randomstring2”)
- Generate HMAC key using HMACEval(Base Key, “randomstring3”)
- Encrypt User struct to create EncryptedUser using SymEnc(SymmKey, iv (random), User)
- Find the HMAC of EncryptedUser by using HMACEval(HMAC key, EncryptedUser)
- Create some value EncUserData = (EncryptedUser, HMAC)
- Store User struct in the Datastore using DatastoreSet(UUID, EncUserData)
- Return pointer to the User struct

GetUser()

- Calculate SymmKey, HMAC key, and UUID as in InitUser()
- Obtain EncUserData by using DatastoreGet(UUID)
- Obtain the original User struct using AuthDec(Base Key, EncUserData)
- Check integrity and authenticate by comparing HMAC with HMACEval(HMAC key, EncryptedUser) using HMACEqual(stored/tag HMAC, calculated HMAC)
    - If fail then error
- Decrypt using SymDec(SymmKey, EncryptedUser)
- Return pointer to the User struct

<u>Warning for Deterministic encoding to a UUID:</u>

The encoding does not hide the information in the byte slice. That is to say, you should not pass a confidential secret key as the byte slice here and use the resulting UUID publicly; an attacker may figure out information about the secret key from that UUID.

StoreFile(filename string, data []byte)

- We will store the file in the DataStore using a key UUID.
    - File is broken up into chunks so each chunk must have its own UUID.
    - UUID is generated with uuid.New()
- We need to store this UUID for the file in the user struct so that we can retrieve the same file for the user later; this is where we use the hash table in the user struct.
- Also generate a random new FileEnc Key from RandomBytes to use for encrypting
- Calculate BaseKey from username and password stored in the struct
- Server cannot know filename so we will generate a key for the hash table with HMACEval; also store the encryption key File Key
    - (Key, Value) = (HMACEval(BaseKey, filename), (UUID to LL, FileEnc Key)))
- This value in the hash table is the UUID to the linked list of UUIDs for the file chunks
- Break the file up into chunks of size (___)
- Chunk encryption:
    - Use AuthEnc(FileEnc, chunk) for each chunk to create EncChunkData
    - Generate a UUID for each data chunk using uuid.New()
    - Store UUID in the Linked List of UUIDS
        - Encrypt UUID using AuthEnc(FileEnc Key, UUID)
    - Store the encrypted chunk in the DataStore using DataStoreSet(UUID, EncChunkData)
- Store the encrypted Linked List in the DataStore using DataStoreSet(UUID, LinkedList)

- For each chunk perform the following authenticated encryption procedure:
    - Generate 16 random bytes which we will use as the “Base Key”
    - Generate a SymmKey for encryption using HMACEval(Base Key, RandomBytes(_))[:16]
    - Generate a HMACKey using HMACEval(Base Key, filename + “chunk<i>”)
    - Encrypt the chunk using SymEnc(SymmKey, RandomBytes(_), chunk) to get EncChunk
    - Generate the HMAC of the chunk,HMACChunk, using HMACEval(HMACKey, EncChunk)
    - Create some value EncChunkData = (EncChunk, HMACChunk)
    - Store chunk using DataStoreSet(UUID, EncChunkData)

Pseudocode for AuthEnc/AuthDec

- AuthEnc(Base Key, Plaintext) returns (Ciphertext, HMAC)
    - Generate SymmKey using HMACEval(Base Key, “randomstring1”)
    - Generate HMAC key using HMACEval(Base Key, “randomstring2”)
    - Encrypt using using SymEnc(SymmKey, iv (random), Plaintext) to get Ciphertext
    - Find the HMAC of Ciphertext by using HMACEval(HMAC key, Ciphertext)
    - Return (Ciphertext, HMAC)
- AuthDec(Base Key, (Ciphertext, HMAC_tag)) returns Plaintext
    - Generate SymmKey using HMACEval(Base Key, “randomstring1”)
    - Generate HMAC key using HMACEval(Base Key, “randomstring2”)
    - Find the HMAC of Ciphertext by using HMACEval(HMAC key, Ciphertext)
    - Check integrity and authenticate by comparing HMAC_tag with HMACEval(HMAC key, Ciphertext) using HMACEqual(HMAC_tag, calculated HMAC)
        - If fail then error
    - Decrypt using using SymDec(SymmKey, Ciphertext) to get Plaintext
    - Return Plaintext

LoadFile(filename String)

- If file filename does not exist return nil data and an error
- If the LL holding the UUIDs is a blank string return error
- R
- un AuthDuec and return results of that function

AppendFile(*filename* string, *data* []byte))

- If file *filename* does not exist return a non-nil error
- To append file, compute the key value associated to file *filename* for FileToUUID and retrieve the UUIDList
- Use DataStoreGet(*UUIDList*[len(*UUIDList*) - 1]) to get the last chunk of file *filename*.
- Perform DataStoreDelete(*UUIDList*[len(*UUIDList*) - 1])
- Use AuthDec on the retrieved chunk
- Append *data* to the chunk to create []byte called *newData.*
- Break *newData* into appropriately sized chunks

- Generate UUIDs for the new chunks. Overwrite the last element of *UUIDList* with the first newly generated UUID and append the remaining to the end of *UUIDListl*.
- Use AuthEnc on the new chunks with BaseKey associated to the file and store the encrypted chunks to the DataStore.
- Upload the updated User struct using AuthEnc to encrypt it.

KEY	Derived From	USAGE
Signing key (sk)	DSKeyGen()	Store in User
Verifying key (vk)	DSKeyGen()	Store in KeyStore using key ____ (example key: “<username>VerifyKey”)
Encryption key (ek)	PKEKeyGen()	Store in KeyStore using key ____ (example key: “<username>EncKey”)
Decryption key	PKEKeyGen()	Store in User
Base Key	Argon2Key(password, username, 16)	The key based on username and password used to derive the 3 other keys for uuid, hmac, and symmkey enc.
UUID (User Struct)	HMACEval(Base Key, “randomstring1”)	Used for the UUID to store User struct in DataStore.
Symmkey (User Struct)	HMACEval(Base Key, “randomstring2”)	Used to encrypt/decrypt the User struct.
Hmac key (User Struct)	HMACEval(Base Key, “randomstring3”)	Used to authenticate the User struct.
FileEnc Key	Random(16)	Used to encrypt the chunks

KEY	Derived From	USAGE

Part 2 Design

If I fell asleep and don't answer I'll try coding this part and just hope part 1 works, tho tbh I think part 1 is much harder to code so good luck :')

ShareFile(filename string, recipient string) (magic_string string, err error)

- This function is called by the owner of the file
- Remember that (Key, Value) = (HMACEval(BaseKey, filename), (UUID to LL, FileEnc Key)))
- Obtain (UUID to LL, FileEnc Key) from the hashmap, call it MagicKey
- Get the public key of the recipient
- Encrypt MagicKey, which equals (UUID to LL, FileEnc Key), using PKEEnc(recipient Pk, MagicKey) to get EncMagicKey
- Create a digital signature Sig for EncMagicKey using owner's signing key (private) through the function DSSign(DSSignKey, EncMagicKey)
- The magic_string is (Sig, EncMagicKey), Sig should be 256 bytes
- Return magic_string

ReceiveFile(filename string, sender string, magic_string string) error

- This function is called by the receiver of the file
- Get sender/owner's verification key from KeyStore using sender username (which is sender string)
- Use sender/owner's verification key to verify the authenticity and integrity of the EncMagicKey in magic_string
- Decrypt EncMagicKey using receiver's private key
- EncMagicKey equals (UUID to LL, FileEnc Key), so store this value into the receiver's hashmap with key HMACEval(BaseKey, filename)
    - We do this since (Key, Value) = (HMACEval(BaseKey, filename), (UUID to LL, FileEnc Key)))
    - If the filename already exists then error (if the same key is already in hashmap)
- Now the receiver has access to the same physical file in the datastore as the sender, so can read/edit/share the file

RevokeFile(filename string) error

- This function is called by the owner of the file
- Reminder that in the hashmap (Key, Value) = (HMACEval(BaseKey, filename), (UUID to LL, FileEnc Key)))
- Obtain (UUID to LL, FileEnc Key) from the key generated from the filename, which is HMACEval(BaseKey, filename)
- Get the encrypted Linked List of UUIDs from the Datastore using DatastoreGet(UUID to LL)
- Delete the list of UUIDs from the current location in DataStore
- Randomly generate a new random UUID using uuid.New()
- Store the encrypted LL in the new UUID

- (possibly we should decrypt and re-encrypt the LL with a new random key)
- Now everyone else we shared the file with will get an error when trying to access it, but we safely stored the LL with the file locations somewhere else in the DataStore and saved its location