Project 2 Autograder

● Graded

4 Days, 17 Hours Late

Group

Shuhua Zhang
Ruihuan Gao
View or edit group

Total Points

83.5715 / 120 pts

Autograder Score

83.5715 / 120.0

Failed Tests

Functionality [InitUser/GetUser] (0/1)
Functionality [RevokeAccess] (0/1)
Efficiency Requirements (0/1)
Security: Confidentiality Requirements (0/1)
Functionality [CreateInvite/AcceptInvite] (0/1)
Security: Integrity Requirements (0/1)

Passed Tests

Design Requirements (1/1)
Functionality [StoreFile/LoadFile/AppendToFile] (1/1)
Basic Tests (1/1)
Formatting Tests (1/1)
Code Coverage Flags (20/20)

Autograder Results

Design Requirements (1/1)

Passed 2/2 test cases

Functionality [InitUser/GetUser] (0/1)

Passed 9/10 test cases

Failed Tests:

[0.01 Penalty] Attempt to create a user that already exists after datastore adversary deletes user from datast

Actual Penalty: 0.01
Maximum Penalty: 0.10
Applied Penalty: 0.01

Functionality [RevokeAccess] (0/1)

Passed 9/10 test cases

Failed Tests:

[0.04 Penalty] A complicated chain of shares & revokes.

Actual Penalty: 0.04
Maximum Penalty: 0.15
Applied Penalty: 0.04

Efficiency Requirements (0/1)

Passed 2/4 test cases

Failed Tests:

[0.08 Penalty] Append should NOT scale with the number of files a user has.
[0.08 Penalty] Append should NOT scale with the length of a user's username.

Actual Penalty: 0.15
Maximum Penalty: 0.10
Applied Penalty: 0.10

Security: Confidentiality Requirements (0/1)

Passed 7/8 test cases

Failed Tests:

[0.08 Penalty] The client should not leak the length of filenames.

Actual Penalty: 0.08
Maximum Penalty: 0.40
Applied Penalty: 0.08

Functionality [CreateInvite/AcceptInvite] (0/1)

Passed 21/23 test cases

Failed Tests:

[0.04 Penalty] Delayed accept: write, create invite bob, create invitation charles, accept invitation charles,revo
[0.01 Penalty] Call AcceptInvitation after swapping invitation with another for a different file.

Actual Penalty: 0.05
Maximum Penalty: 0.15
Applied Penalty: 0.05

Functionality [StoreFile/LoadFile/AppendToFile] (1/1)

Passed 12/12 test cases

Basic Tests (1/1)

Passed 4/4 test cases

Formatting Tests (1/1)

Passed 2/2 test cases

Security: Integrity Requirements (0/1)

Passed 22/24 test cases

Failed Tests:

[0.08 Penalty] Fuzz Testing: Two users. Test user struct integrity while swapping datastore entries.
[0.08 Penalty] Fuzz Testing: Two users. Test user struct integrity while copying datastore entries into different

Actual Penalty: 0.15
Maximum Penalty: 0.15
Applied Penalty: 0.15

Code Coverage Flags (20/20)

The student written tests achieved the following code coverage flags when run against the staff solution:
0 1 2 3 4 5 6 7 8 9 10 14 15 17 18 19 20 22 27 28 31 33

Submitted Files

.gitignore Download

```
# Created by https://www.toptal.com/developers/gitignore/api/go,goland,visualstudiocode,sublimetext
# Edit at https://www.toptal.com/developers/gitignore?templates=go,goland,visualstudiocode,sublimetext

### Go ###
# If you prefer the allow list template instead of the deny list, see community template:
# https://github.com/github/gitignore/blob/main/community/Golang/Go.AllowList.gitignore
#
# Binaries for programs and plugins
*.exe
*.exe~
*.dll
*.so
*.dylib

# Test binary, built with `go test -c`
*.test

# Output of the go coverage tool, specifically when used with LiteIDE
*.out

# Dependency directories (remove the comment below to include it)
# vendor/

# Go workspace file
go.work

### GoLand ###
# Covers JetBrains IDEs: IntelliJ, RubyMine, PhpStorm, AppCode, PyCharm, CLion, Android Studio, WebStorm and Rider
# Reference: https://intellij-support.jetbrains.com/hc/en-us/articles/206544839

# User-specific stuff
.idea/**/workspace.xml
.idea/**/tasks.xml
.idea/**/usage.statistics.xml
.idea/**/dictionaries
.idea/**/shelf

# AWS User-specific
.idea/**/aws.xml

# Generated files
.idea/**/contentModel.xml

# Sensitive or high-churn files
.idea/**/dataSources/
.idea/**/dataSources.ids
.idea/**/dataSources.local.xml
```

```
.idea/**/sqlDataSources.xml
.idea/**/dynamic.xml
.idea/**/uiDesigner.xml
.idea/**/dbnavigator.xml

# Gradle
.idea/**/gradle.xml
.idea/**/libraries

# Gradle and Maven with auto-import
# When using Gradle or Maven with auto-import, you should exclude module files,
# since they will be recreated, and may cause churn.  Uncomment if using
# auto-import.
# .idea/artifacts
# .idea/compiler.xml
# .idea/jarRepositories.xml
# .idea/modules.xml
# .idea/*.iml
# .idea/modules
# *.iml
# *.ipr

# CMake
cmake-build-*/

# Mongo Explorer plugin
.idea/**/mongoSettings.xml

# File-based project format
*.iws

# IntelliJ
out/

# mpeltonen/sbt-idea plugin
.idea_modules/

# JIRA plugin
atlassian-ide-plugin.xml

# Cursive Clojure plugin
.idea/replstate.xml

# SonarLint plugin
.idea/sonarlint/

# Crashlytics plugin (for Android Studio and IntelliJ)
com_crashlytics_export_strings.xml
crashlytics.properties
crashlytics-build.properties
fabric.properties

```

```
# Editor-based Rest Client
.idea/httpRequests

# Android studio 3.1+ serialized cache file
.idea/caches/build_file_checksums.ser

### GoLand Patch ###
# Comment Reason: https://github.com/joeblau/gitignore.io/issues/186#issuecomment-215987721

# *.iml
# modules.xml
# .idea/misc.xml
# *.ipr

# Sonarlint plugin
# https://plugins.jetbrains.com/plugin/7973-sonarlint
.idea/**/sonarlint/

# SonarQube Plugin
# https://plugins.jetbrains.com/plugin/7238-sonarqube-community-plugin
.idea/**/sonarIssues.xml

# Markdown Navigator plugin
# https://plugins.jetbrains.com/plugin/7896-markdown-navigator-enhanced
.idea/**/markdown-navigator.xml
.idea/**/markdown-navigator-enh.xml
.idea/**/markdown-navigator/

# Cache file creation bug
# See https://youtrack.jetbrains.com/issue/JBR-2257
.idea/$CACHE_FILE$

# CodeStream plugin
# https://plugins.jetbrains.com/plugin/12206-codestream
.idea/codestream.xml

# Azure Toolkit for IntelliJ plugin
# https://plugins.jetbrains.com/plugin/8053-azure-toolkit-for-intellij
.idea/**/azureSettings.xml

### SublimeText ###
# Cache files for Sublime Text
*.tmlanguage.cache
*.tmPreferences.cache
*.stTheme.cache

# Workspace files are user-specific
*.sublime-workspace

# Project files should be checked into the repository, unless a significant
# proportion of contributors will probably not be using Sublime Text
# *.sublime-project
```

```
152
153 # SFTP configuration file
154 sftp-config.json
155 sftp-config-alt*.json
156
157 # Package control specific files
158 Package Control.last-run
159 Package Control.ca-list
160 Package Control.ca-bundle
161 Package Control.system-ca-bundle
162 Package Control.cache/
163 Package Control.ca-certs/
164 Package Control.merged-ca-bundle
165 Package Control.user-ca-bundle
166 oscrypto-ca-bundle.crt
167 bh_unicode_properties.cache
168
169 # Sublime-github package stores a github token in this file
170 # https://packagecontrol.io/packages/sublime-github
171 GitHub.sublime-settings
172
173 ### VisualStudioCode ###
174 .vscode/*
175 !.vscode/settings.json
176 !.vscode/tasks.json
177 !.vscode/launch.json
178 !.vscode/extensions.json
179 !.vscode/*.code-snippets
180
181 # Local History for Visual Studio Code
182 .history/
183
184 # Built Visual Studio Code Extensions
185 *.vsix
186
187 ### VisualStudioCode Patch ###
188 # Ignore all local history of files
189 .history
190 .ionide
191
192 问题
193
194 # End of https://www.toptal.com/developers/gitignore/api/go,goland,visualstudiocode,sublimetext
```

CHANGELOG.md Download

```
# Changelog
All notable changes to this project will be documented in this file.

The format is based on [Keep a Changelog][Keep a Changelog] and this project adheres to [Semantic Versioning][Semantic Versioning].

## [Unreleased]

## [v0.2.0] - 2021-03-29
### Changed
- Updated [userlib][userlib] dependency to `v0.2.0`.

---

## [Released]

## [v0.1.0] - 2021-02-21
CHANGELOG did not exist in this release.

---

<!-- Links -->
[Keep a Changelog]: https://keepachangelog.com/
[Semantic Versioning]: https://semver.org/
[userlib]: https://github.com/cs161-staff/project2-userlib/blob/master/CHANGELOG.md

<!-- Versions -->
[Unreleased]: https://github.com/cs161-staff/project2-starter-code/compare/v0.2.0...HEAD
[Released]: https://github.com/cs161-staff/project2-starter-code/releases
[v0.2.0]: https://github.com/cs161-staff/project2-starter-code/compare/v0.1.0...v0.2.0
[v0.1.0]: https://github.com/cs161-staff/project2-starter-code/releases/v0.1.0

```

Design Doc.md Download

````
### **Design Question: Data structures（不确定）**

```go

type UserMetadata struct {
    EncryptedPrivateKey []byte   // User's encrypted private key
    PublicKey           []byte   // Public key for encryption
    SignatureKey        []byte   // Public key for verifying digital signatures
    RootFilePointer     []byte   // Pointer to encrypted root directory
    FileMappings        map[string]UUID // Local filename to File UUID mapping
}

type File struct {
    VersionVector      map[string]int // Track versions per device
    Owner              string
    EncryptedFileKey   []byte
    OwnerSignature     []byte
    SharedWith         map[string][]byte // Encrypted file keys for each user
    CRDT_Log           []LogEntry
}

type FileContentBlock struct {
    EncryptedChunk  []byte
    ChunkNumber     int
    HMACTag         []byte
    Timestamp       int64 // Used for conflict resolution
    DeviceID        string // Identifies the uploader
}

type Invitation struct {
    FilePointer    []byte   // Encrypted reference to shared file metadata
    EncryptedKey   []byte   // Symmetric key encrypted for recipient
    SenderSignature []byte  // Ensures invitation integrity
}

```

---
### **Design Question: Datastore Adversary**

#### Storing a Hashed Password in Datastore Won't Work

1. Datastore adversaries can read H(password) or overwrite H(password) with H(their_password), allowing them to log in as the user.
2. Even if login authentication is correct, the adversary can still read, modify, or delete other stored files in Datastore if they are not securely encrypted.
````

3. If the encryption key is stored in Datastore unprotected, an attacker can extract it and decrypt all user files.

Secure Design for Storing Information in Datastore

1. Password-Derived Key (PDK) Generation

Instead of storing a hashed password, use argon2Key(password, salt, keyLen) (有对应helper function) to generate the master key.

Alternative Method 1: use salted hash:
```go
PDK = H(password || salt). For each user, store username, salt, PDK.
```

Alternative Method 2: use PBKDF2 (似乎比较难实现，lec12-P28) to generate a key from the password:
```go
PDK = PBKDF2 (password || salt).
```

Salt is unique, long and random.
From master key, generate keys, e.g. encryption key(enc_*key), HMAC key(mac*_key).

2. Encrypted User Metadata

We store encrypted user metadata instead of plaintext authentication information. enc_*key and mac*_key are not stored.

```go
ciphertext = AES_CTR_Enc(enc_key, plaintext) (有对应helper function)

mac = HMAC-SHA256(mac_key, ciphertext)  (有对应helper function)
```

Alternative Method:
```go
EncryptedMetadata = AES_GCM_Enc(PDK, plaintext) (math? lec9-P38)
```

3. File Encryption & Secure Storage (不确定)

File contents are encrypted with a secure random symmetric key before storing:
``` go
FileKey = SecureRandom(32)
EncryptedFileContent = AES_GCM_Enc(FileKey, FileContent)
```
File metadata (which contains FileKey) is encrypted with the user's public key.
``` go

symKey := SecureRandom(32) // 生成安全的对称密钥
encryptedSymKey, err := PKE_Encrypt(UserPublicKey, symKey)
encryptedData := AES_GCM_Encrypt(symKey, FileContent) // 使用对称密钥加密文件数据
```

```

##### 4. Integrity Protection Against Tampering (不确定)

To prevent an attacker from modifying stored data, we use HMAC or Digital Signatures:
Every stored object has a signature:
```go
IntegritySignature = Sign(UserPrivateKey, Data)
```
Upon retrieval, the signature is verified:
```go
Verify(UserPublicKey, Data, IntegritySignature)
```
If the adversary modifies anything, the signature check will fail, and the system will detect tampering.

Design Question: Helper functions

1. Authenticated Encryption Helper

Combines encryption and integrity protection.

Function: AuthenticatedEncrypt
Purpose: Encrypt data and generate an integrity tag (HMAC).
Input:
- key: Symmetric key for encryption.
- iv: Initialization vector (IV) for AES-CTR.
- plaintext: Data to encrypt.
Output:
- ciphertext: Encrypted data.
- hmacTag: HMAC tag for integrity verification.

```go
func AuthenticatedEncrypt(key []byte, iv []byte, plaintext []byte) (ciphertext []byte, hmacTag []byte, err error) {
    // Encrypt the plaintext using AES-CTR
    ciphertext = SymEnc(key, iv, plaintext)

    // Generate HMAC for integrity
    hmacTag, err = HMACEval(key, ciphertext)
    if err != nil {
        return nil, nil, err
    }

    return ciphertext, hmacTag, nil
}

// Function: AuthenticatedDecrypt
// Purpose: Decrypt data and verify integrity.
// Input:
// - key: Symmetric key for decryption.
```
```

```
// - ciphertext: Encrypted data.
// - hmacTag: HMAC tag for integrity verification.
// Output:
// - plaintext: Decrypted data.
// - err: Error if HMAC verification fails.
func AuthenticatedDecrypt(key []byte, ciphertext []byte, hmacTag []byte) (plaintext []byte, err error) {
    // Verify HMAC
    computedHmac, err := HMACEval(key, ciphertext)
    if err != nil {
        return nil, err
    }

    if !HMACEqual(computedHmac, hmacTag) {
        return nil, errors.New("HMAC verification failed: data tampered")
    }

    // Decrypt the ciphertext using AES-CTR
    plaintext = SymDec(key, ciphertext)
    return plaintext, nil
}
```

2. Hybrid Encryption Helper

Combines symmetric encryption (for efficiency) with public-key encryption (for secure key exchange)

Function: HybridEncrypt
Purpose: Use the given public key to encrypt a random symmetric key. Then, use the symmetric key to encrypt the actual data.
Input:
- publicKey: Recipient's public key (PKEEncKey).
- plaintext: Data to encrypt.
Output:
- encryptedSymKey: Encrypted data.symmetric key (encrypted with the public key)
- encryptedData: data (encrypted with the symmetric key)
- err: Error if encryption fails.

```go

func HybridEncrypt(publicKey PKEEncKey, plaintext []byte) (encryptedSymKey []byte, encryptedData []byte, err error) {
    // Generate a random symmetric key
    symKey := RandomBytes(AESKeySizeBytes)

    // Encrypt the symmetric key using the recipient's public key
    encryptedSymKey, err = PKEEnc(publicKey, symKey)
    if err != nil {
        return nil, nil, err
    }

    // Encrypt the plaintext using the symmetric key
```

```
    iv := RandomBytes(AESBlockSizeBytes)
    encryptedData = SymEnc(symKey, iv, plaintext)

    return encryptedSymKey, encryptedData, nil
}

// Function: HybridDecrypt
// Purpose: Use the given private key to decrypt the symmetric key.
// Then, use the symmetric key to decrypt the data.
// Input:
// - privateKey: Recipient's private key (PKEDecKey).
// - encryptedSymKey: Encrypted symmetric key.
// - encryptedData: Encrypted data.
// Output:
// - plaintext: Decrypted data.
// - err: Error if decryption fails.
func HybridDecrypt(privateKey PKEDecKey, encryptedSymKey []byte, encryptedData []byte) (plaintext
[]byte, err error) {
    // Decrypt the symmetric key using the recipient's private key
    symKey, err := PKEDec(privateKey, encryptedSymKey)
    if err != nil {
        return nil, err
    }

    // Decrypt the ciphertext using the symmetric key
    plaintext = SymDec(symKey, encryptedData)
    return plaintext, nil
}

```
3. Key Derivation Helper

Generate keys from a password or master key.

Function: DeriveKeys
Purpose: Derive multiple keys (e.g., encryption key, HMAC key) from a master key.
Input:
- masterKey: source key (e.g., password-derived key).
- context: Context string.
Output:
- keys: Derived keys (e.g., encryption key, HMAC key).
- err: Error if derivation fails.
```go
func DeriveKeys(masterKey []byte, context string) (keys map[string][]byte, err error) {
    keys = make(map[string][]byte)

    // Derive encryption key
    encKey, err := HashKDF(masterKey, []byte("encryption"))
    if err != nil {
        return nil, err
    }
    keys["encryption"] = encKey[:AESKeySizeBytes]
```

```
248
249     // Derive HMAC key
250     hmacKey, err := HashKDF(masterKey, []byte("hmac"))
251     if err != nil {
252         return nil, err
253     }
254     keys["hmac"] = hmacKey[:AESKeySizeBytes]
255
256     return keys, nil
257 }
258
259 ```
260
261 ##### 4. File Metadata Helper （不確定）
262
263 Store and retrieve file metadata (e.g., owner, file ID) securely.
264
265 **Function:** StoreFileMetadata
266 **Purpose:** Store file metadata with integrity protection.
267 **Input:**
268 - fileID: UUID of the file.
269 - metadata: File metadata (e.g., owner, file key).
270 - signingKey: Private key for signing metadata.
271 **Output:**
272 - err: Error if storage fails.
273 ```go
274 // StoreFileMetadata stores the file metadata and signs it.
275 func StoreFileMetadata(fileID UUID, metadata []byte, signingKey DSSignKey) error {
276     // Sign the metadata
277     signature, err := DSSign(signingKey, metadata)
278     if err != nil {
279         return err
280     }
281
282     // Combine metadata and signature
283     data := append(metadata, signature...)
284
285     // Store in Datastore
286     DatastoreSet(fileID, data)
287     return nil
288 }
289
290 // Function: LoadFileMetadata
291 // Purpose: Load and verify file metadata.
292 // Input:
293 // - fileID: UUID of the file.
294 // - verificationKey: Public key for verifying metadata.
295 // Output:
296 // - metadata: File metadata.
297 // - err: Error if verification fails.
298 func LoadFileMetadata(fileID UUID, verificationKey DSVerifyKey) (metadata []byte, err error) {
299     // Load metadata and signature from Datastore
```

```
    data, ok := DatastoreGet(fileID)
    if !ok {
        return nil, errors.New("file metadata not found")
    }

    // Split metadata and signature
    metadata = data[:len(data)-HashSizeBytes]
    signature := data[len(data)-HashSizeBytes:]

    // Verify signature
    err = DSVerify(verificationKey, metadata, signature)
    if err != nil {
        return nil, errors.New("metadata verification failed")
    }

    return metadata, nil
}
```

Design Question: User Authentication

InitUser():

1. Check if a record for the username already exists in the Datastore. If it exists, return an error.
2. Use Argon2Key to derive a master key from the password and salt. The salt must be unique and random.
3. Use DeriveKeys to derive an encryption key (enc_*key) and an HMAC key (mac*_key) from the master key.
4. Derive the encrypted user metadata with enc_*key. Generate HMAC tag with encrypted user metadata and mac*_key.
5. Store the salt, encrypted user metadata, and HMAC tag in the Datastore. Use HMAC to protect the integrity of user information.
6. Create a User object in memory and return its pointer

GetUser():

1. Check if the username exists in the Datastore. If it doesn't exist, return an error.
2. Verify the password:

- Retrieve the salt and encrypted user metadata from Datastore.
- Use the input password and salt to derive the new master key.
- Use the new master key to derive the encryption key (enc_*key2)and HMAC key (mac*_key2).
- Generate HMAC tag2 with encrypted user metadata and mac_key2.
- Verify the HMAC tag2 with HMAC tag. If yes, decrypt the user metadata.

3. If the password is correct and the data is intact, create and return the User object. Otherwise, return an error.

If the password is incorrect, it will not pass the HMAC tag verification.

Design Question: Multiple Devices（不确定）

User objects do not store file data in local memory; they only hold keys and necessary metadata. All file operations directly read/write from Datastore.
File Content stored in chunks, each with a unique UUID. File metadata maintains a list of chunks and a version number to ensure atomic updates.
AppendToFile
1. Read the current metadata and version number.
2. Generate a new chunk and store it
3. Update the chunk list and version number
4. Check if the versionvector number is unchanged:
5. If unchanged, write the new metadata and increment the version number.
6. If changed, retry the entire operation (similar to optimistic locking).

```go
func AppendToFile(filename string, content []byte, userMetadata UserMetadata) error {
    for {
        // 1. Get current file metadata and version vector
        file, err := loadFile(filename)
        if err != nil {
            return fmt.Errorf("failed to load file metadata: %w", err)
        }

        // 2. Get user's version vector entry for conflict resolution
        userVersion, exists := file.VersionVector[userMetadata.PublicKey]
        if !exists {
            userVersion = 0
        }

        // 3. Decrypt the symmetric file key
        symKey, err := decryptSymKey(file.EncryptedFileKey, userMetadata.EncryptedPrivateKey)
        if err != nil {
            return fmt.Errorf("failed to decrypt symmetric key: %w", err)
        }

        // 4. Encrypt the new content chunk
        encryptedContent, err := encrypt(content, symKey)
        if err != nil {
            return fmt.Errorf("failed to encrypt content: %w", err)
        }

        // 5. Generate HMAC tag for the chunk to ensure integrity
        hmacTag, err := generateHMAC(encryptedContent, symKey)
        if err != nil {
            return fmt.Errorf("failed to generate HMAC: %w", err)
        }

        // 6. Create a new FileContentBlock
        newBlock := FileContentBlock{
            EncryptedChunk: encryptedContent,
            ChunkNumber:    len(file.CRDT_Log) + 1,
            HMACTag:        hmacTag,
```

```
        }

        // 7. Store the new block in Datastore
        newBlockID := uuid.New()
        DatastoreSet(newBlockID, newBlock)

        // 8. Update the file metadata
        file.VersionVector[userMetadata.PublicKey] = userVersion + 1 // Increment user's version entry
        newLogEntry := LogEntry{
            Timestamp: time.Now(),
            Action:    "Append",
            User:      userMetadata.PublicKey,
            ChunkID:   newBlockID,
        }
        file.CRDT_Log = append(file.CRDT_Log, newLogEntry) // Add operation to CRDT log

        // 9. Sign the updated metadata
        signatureData := generateSignatureData(file)
        file.OwnerSignature, err = signMetadata(signatureData, userMetadata.EncryptedPrivateKey)
        if err != nil {
            return fmt.Errorf("failed to sign metadata: %w", err)
        }

        // 10. Compute HMAC for metadata integrity
        hmacNew := HMAC_SHA256(symKey, file)

        // 11. Attempt atomic update using Compare-and-Swap with HMAC verification
        success := compareAndSwapWithHMAC(filename, file, hmacNew)
        if success {
            return nil // Success, exit the function
        }

        // Version conflict detected, retry operation
        log.Println("Version conflict detected, retrying append operation...")
        retryAppend()
    }
}

```


Design Question: File Storage and Retrieval

Overview

This design ensures confidentiality, integrity, and user isolation in file storage and retrieval. File contents, filenames, and filename lengths remain private, and any tampering is detectable.

File Storage

1. **Encryption and Integrity Protection**

448   - A **unique symmetric key** is generated for each file and used for **AES-GCM encryption**.
449   - An **HMAC** is computed over the encrypted file to ensure integrity.
450 2. **Secure Filename Handling**
451
452   - The filename is **encrypted** with a user-specific key.
453   - The **encrypted filename is hashed** to a fixed-length identifier, preventing leaks about its content or length.
454 3. **Data Storage**
455
456   - The encrypted file is stored in chunks, each assigned a **random UUID**.
457   - Chunks are stored in the **datastore** using their UUIDs.
458   - Metadata, including UUIDs and the **encrypted symmetric key**, is stored under a **user-specific key**.
459
460 #### **File Retrieval**
461
462 1. **Retrieve Metadata**
463
464   - EvanBot provides the filename.
465   - The filename is **encrypted and hashed**, and the metadata is retrieved using this hash.
466 2. **Decrypt the Symmetric Key**
467
468   - The **symmetric key is decrypted** using EvanBot's **private key**.
469 3. **Retrieve and Verify Chunks**
470
471   - File chunks are fetched using their UUIDs.
472   - The **HMAC is verified** to ensure integrity.
473 4. **Decrypt and Reassemble File**
474
475   - If the HMAC check passes, chunks are **decrypted and reassembled** into the original file.
476
477 #### **Security Considerations**
478
479 - **Confidentiality**: File contents and filenames are always encrypted.
480 - **Filename Protection**: Filenames are encrypted and hashed, preventing exposure.
481 - **UUIDs for Storage**: Randomly generated UUIDs prevent pattern recognition.
482 - **Integrity Verification**: HMAC ensures files have not been tampered with.
483 - **User Isolation**: Even if different users store files with the same name, they remain private and separate.
484
485 ### **Design Question: Efficient Append**
486
487 To ensure an efficient append operation while minimizing bandwidth consumption, the following optimized process is proposed:
488 #### **1. Retrieve File Metadata:**
489 - **Fetch Metadata:** Retrieve the user's metadata to obtain the File UUID and the encrypted symmetric key for the file.
490 - **Decrypt Symmetric Key:** Use the user's private key to decrypt the symmetric key securely.
491 #### **2. Append New Data:**
492 - **Encrypt New Data:** Encrypt the new data chunk using the symmetric key before storage.
493 - **Compute HMAC:** Generate the HMAC for the new encrypted chunk to ensure integrity.
494 - **Generate New Chunk UUID:** Assign a unique identifier (UUID) for the new data chunk.

- **Store New Chunk:** Upload the encrypted chunk and its corresponding HMAC to the Datastore under the new Chunk UUID.
3. Update File Metadata:
- **Fetch Current Metadata:** Retrieve the latest metadata associated with the File UUID.
- **Update Chunk List:** Append the new Chunk UUID to the list of existing chunks in the file's metadata.
- **Increment Version Number:** Increase the file's version number to reflect the append operation.
- **Compute Metadata HMAC:** Generate an HMAC for the updated metadata to ensure integrity.
- **Store Updated Metadata:** Upload the revised metadata along with its HMAC back to the Datastore under the File UUID.

Bandwidth Analysis:
Upload (DatastoreSet):
- **New Encrypted Chunk:** The size equals the size of the newly appended data (**|append_data|**).
- **Chunk HMAC:** A fixed size, typically 32 bytes.
- **Updated Metadata:** Includes the revised list of chunk UUIDs and the incremented version number.
- **Metadata HMAC:** A fixed size, typically 32 bytes.

```markdown
|append_data| + 32 bytes (chunk HMAC) + size of updated metadata + 32 bytes (metadata HMAC)
```

Download (DatastoreGet):
- **User Metadata:** Required to retrieve the encrypted symmetric key and current file metadata.
- **File Metadata:** Necessary to obtain the latest list of chunk UUIDs and the version number.
```markdown
size of user metadata + size of file metadata
```

This approach ensures that the total bandwidth usage scales **linearly** with the size of the appended data (**|append_data|**), while the overhead from metadata and HMACs remains **constant and minimal**. The design guarantees bandwidth efficiency by eliminating unnecessary data retrievals and uploads, focusing only on the new data and its corresponding metadata updates.

Design Question:File Sharing
1. CreateInvitation - What Gets Created in Datastore?
When **EvanBot (file owner) shares a file with CodaBot**, the system stores an **invitation** in Datastore.
1. **Fetch File Metadata**
- Retrieve **FileUUID** and **EncryptedFileKey**.
2. **Encrypt File Key for CodaBot**
- Encrypt the file's symmetric key with **CodaBot's public key**:
3. **Create and Store Invitation**
- Generate and store an **Invitation struct** in Datastore:
- Store **InvitationUUID** in Datastore and return it to EvanBot.
4. **Send InvitationUUID to CodaBot** (via external channel).
2. AcceptInvitation - What Changes in Datastore?
When **CodaBot accepts the invitation**, it must gain access to the file.
1. **Retrieve and Verify Invitation**

- Fetch `Invitation` from Datastore using `InvitationUUID`.
- Verify `SenderSignature` to ensure authenticity:
2. **Decrypt File Key**
- CodaBot decrypts the **file symmetric key** using its **private key**:
3. **Update CodaBot's Metadata**
- Store `FileUUID` and `decryptedFileKey` in **CodaBot's encrypted UserMetadata**.
4. **Update File's SharedWith Mapping**
- Add CodaBot to `File.SharedWith`:
- Store the updated file metadata in Datastore.
3. How CodaBot Accesses the File in the Future?
- Retrieve **FileUUID** from UserMetadata.
- Fetch **EncryptedFileKey** from `File.SharedWith`.
- Decrypt the **FileKey** using its **private key**.
- Use **FileKey** to decrypt the file's encrypted data.
4. Non-Owner Sharing (CodaBot → PintoBot)
CodaBot (non-owner) shares the file with PintoBot similarly:
1. **Retrieve Encrypted File Key from SharedWith**
- CodaBot fetches **its own encrypted file key** from `SharedWith`.
2. **Encrypt File Key for PintoBot**
- Encrypt the file key with **PintoBot's public key**: ```
3. **Create and Store Invitation**
- Store a new `Invitation` in Datastore.
4. **When PintoBot Accepts:**
- Fetch and verify the invitation.
- Decrypt the **file key**.
- Store `FileUUID` in **PintoBot's UserMetadata**.
- Update `File.SharedWith[PintoBot] = encryptedKeyForPintoBot`.
- Save updated metadata in Datastore.
**Design Question:File Revocation

1. What Happens When A Revokes B's Access?
- **B, D, E, and F lose access.**
- **C and G retain access.**
- **B cannot decrypt the file or track future updates.**
2. Values Updated in Datastore
1. **Remove B from `SharedWith`**
 - B can no longer retrieve the encrypted file key.
2. **Generate a New File Key**
- Prevents B from using any previously obtained key.
3. **Re-Encrypt File Data & Metadata**
- Encrypt file contents with `newFileKey`.
- Re-encrypt `newFileKey` for C, G, and any remaining authorized users.
4. **Invalidate B's Metadata Reference**
- Remove file UUID from B's metadata.
- Ensures B cannot list or reference the file.
3. Preventing Access by a Revoked User
1. **Key Freshness**
- All chunks re-encrypted with `newFileKey`.
- Old keys are now useless.
2. **Prevent Datastore Replay Attacks**
- Store a **version counter** in metadata.
- Ensure every read checks the latest version.

```
3. **Obfuscate Update Timing**
- Introduce **randomized dummy writes** to datastore.
- Prevents B from detecting when the revocation happened.
#### 4. Recursive Revocation (Propagating to Shared Users)
- If B had shared the file, **D, E, and F also lose access**.
- Use **Breadth-First Search (BFS)** to revoke all sub-users.
```go
func RevokeAccess(filename string, targetUser string) {
    queue := []string{targetUser}

    for len(queue) > 0 {
        user := queue[0]
        queue = queue[1:]
        delete(File.SharedWith, user)

        if subUsers, exists := File.SharedWith[user]; exists {
            queue = append(queue, subUsers...)
        }
    }
}
```
5. Security Guarantees

Threat	**Mitigation**
B accesses old datastore values	Fresh encryption keys prevent decryption.
B replays old metadata	Version counter ensures latest state is used.
B detects file updates	Dummy writes and padding prevent timing inference.
B modifies file undetected	Integrity checks (HMAC/signatures) detect tampering.

Final Revocation Steps

1. **Remove B from `SharedWith`.**
2. **Propagate revocation to B's shared users.**
3. **Generate a fresh file key.**
4. **Re-encrypt all file chunks.**
5. **Re-encrypt and store the new key for C, G, etc.**
6. **Update metadata and store securely.**

This ensures **security, efficiency, and minimal metadata exposure.** 🚀
```

Design Doc_revised.md

Download

````
## ** 🔹 Data Structure Design**

### **UserMetadata**

```go
type UserMetadata struct {
    EncryptedPrivateKey []byte    // Encrypted user private key storage
    PublicKey []byte              // User public key
    SignatureKey []byte           // User signature key
    RootFilePointer []byte        // Pointer to the root of the user's file system
    FileMappings map[string]UUID  // Mapping of file names to file UUIDs

}
```

Stores encrypted user keys and file mappings to ensure secure access


### **File - Logical Layer**

```go
type File struct {
    VersionVector map[string]int  // Version control (supports CRDT or concurrent editing)
    Owner string                  // File owner
    EncryptedFileKey []byte       // `FileKey`, used to decrypt `ChunkKeys`
    // Encrypted with the owner's public key (for self-use only)
    OwnerSignature []byte         // File owner's signature to prevent tampering
    SharedWith map[string][]byte  // Shared user list (User → EncryptedFileKeyForUserx)
    CRDT_Log  []LogEntry          // Maintains conflict resolution logs

}
```

Manages access control & sharing information, does not store ``ChunkList``
`SharedWith`supports non-owner sharing and cascading revocation


### **FileMetadata - Physical Storage Layer**

```go
type FileMetadata struct {
    Version int                   // Version number to prevent rollback attacks
    ChunkList []UUID              // List of UUIDs, each corresponding to a FileChunk
    EncryptedChunkKeys map[string][]byte  // Each chunk may have a different key (UUID → EncryptedChunkKey)
    EncryptedFileKey []byte       // `FileKey`, used to decrypt `ChunkKeys`
    // Encrypted with the owner's public key (for self-use only)
    OwnerSignature []byte         // File owner's signature to prevent tampering
    HMAC []byte                   // Ensures metadata integrity
````
````

```
}
```

`EncryptedChunkKeys` changed from an array to a `map[string][]byte`to support `ChunkKey` rotation
Each ``Chunk`` has a separate ``ChunkKey``, ensuring ``ChunkKey`` expiration mechanisms

FileChunk - Data Storage Layer

```go
type FileChunk struct {
    ChunkUUID string              // Unique identifier
    EncryptedData []byte          // AES-CTR encrypted chunk data
    EncryptedChunkKey []byte      // `ChunkKey` encrypted with `FileKey`
    HMACTag []byte                // Integrity check
    KeyExpirationTime int64       // Expiration timestamp (Unix Time)
    ChunkNumber int               // Sequential number
    DeviceID string               // Source device
    PreviousChunkHash []byte      // Optional field to verify sequential integrity
}
```

Each ``Chunk`` has an independent ``ChunkKey`` with a ``KeyExpirationTime`` field.
Even if a ``ChunkKey`` is leaked, access is limited to a short period, requiring re-encryption after expiration.

Invitation - File Sharing

```go
type Invitation struct {
    FilePointer []byte       // UUID pointing to the shared file
    EncryptedKeyForUserx []byte      // `FileKey` encrypted with the receiver's public key
    SenderSignature []byte    // Signed with the sender's private key to prevent tampering
}
```

During sharing, ``EncryptedKeyForUserx`` ensures only the intended receiver can access it, preventing unauthorized access.
Upon revocation, the ``Invitation`` is immediately invalidated and the Receiver can no longer decrypt the ``FileKey``.

Design Plan: Efficient Append

1️⃣ Check if ``FileMetadata.Version`` matches

- **If ``Version`` does not match, re-fetch `FileMetadata`.**

```go
if CachedVersion < FileMetadata.Version:
    ForceRefreshMetadata()
```

```

Ensures append operation is based on the latest `FileMetadata`.


### **2️⃣ Decrypt `FileKey`, then decrypt `ChunkKeys`**
- f the file is not shared, use `ownerPrivateKey` to decrypt `EncryptedFileKey` and ensure the `Version` is valid.
- If the file is shared, use `ReceiverPrivateKey` to decrypt `EncryptedFileKeyForUserX` and ensure the `Version` is valid.
- Decrypt `EncryptedChunkKeys` and check the `Version`

```go
(FileKey, DecryptedVersion) = PKEDec(ReceiverPrivateKey, EncryptedFileKeyForUserX)
if DecryptedVersion < FileMetadata.Version:
    return Error("Old version detected, access denied.")
```

Version check ensures that old `FileKey` cannot access the new data

3️⃣ Generate new `ChunkKey` and encrypt the data

- Generate a new `ChunkKey` for the content to be appended.
- Use `AES-CTR(ChunkKey, append_data) `to encrypt the data.


```plaintext
NewChunkKey = AES-GenerateKey()
EncryptedChunk = AES-CTR(NewChunkKey, append_data)
```

`ChunkKey` is generated randomly, ensuring data independence and security.

4️⃣ Calculate `HMAC`

- Calculate `HMAC(ChunkData || FileMetadata.Version || ChunkUUID)`

```plaintext
HMAC_Chunk = HMAC_SHA256(NewChunkKey, (EncryptedChunk || FileMetadata.Version || ChunkUUID))
```

`HMAC` binds the `Version`, ensuring old keys are invalid

5️⃣ Update `FileMetadata`

- Append `ChunkUUID` to the list.
- Re-encrypt `ChunkKey` and store it in `EncryptedChunkKeys`.
```

```plaintext
EncryptedChunkKey = AES-Encrypt(NewFileKey, (NewChunkKey || FileMetadata.Version))
FileMetadata.ChunkList.append(NewChunkUUID)
FileMetadata.EncryptedChunkKeys[NewChunkUUID] = EncryptedChunkKey
```

The append operation only updates the metadata and `ChunkKey`, without re-encrypting the original data.

6️⃣ Update `FileMetadata.Version` and `HMAC`

- Increment `Version++` to ensure old `ChunkKeys` cannot decrypt the new data.
- Recalculate `HMAC(FileMetadata)` to prevent tampering.

```plaintext
FileMetadata.Version++
FileMetadata.HMAC = HMAC_SHA256(NewFileKey, UpdatedFileMetadata)
```

Changing the `Version` ensures all old data becomes invalid.

**7️⃣ Execute `DummyChunk` Random Write (Enhanced Version)

🔹 New design: **When writing `DummyChunk`, add the `IsDummy = true` flag and avoid adding it to the `ChunkList`.**

🔹 DummyChunk

1. **50% chance to generate `DummyChunk`**
2. `DummyChunk` data structure should include:
   - `IsDummy = true` flag
   - Fake `EncryptedData`
   - Randomly generated `HMAC`
3. `DummyChunk` will not be added to `FileMetadata.ChunkList`, and `FileMetadata.Version` will not be updated.

```go
If Random(0, 1) == 1:
    DummyChunkUUID = GenerateUUID()
    DummyChunkData = RandomData()
    DummyChunk = {
        "EncryptedData": AES-CTR(NewDummyChunkKey, DummyChunkData),
        "IsDummy": true,
        "HMAC": HMAC_SHA256(NewDummyChunkKey, DummyChunkData)
    }
    Store DummyChunk to Datastore  // ❗ DummyChunk do not update ChunkList
```

Valid `ChunkData` will still update `ChunkList`, while `DummyChunk` only disguises itself as a real write, misleading attackers about the update timing.

Shared Design Scheme

1️⃣ `CreateInvitation(sender, receiver, fileUUID)`
Goal: Generate a sharing invitation and encrypt the FileKey for secure sharing.

Steps
1. **Obtain `FileUUID`**
2. **Decrypt `EncryptedFileKey` using `OwnerPrivateKey` to get `FileKey`**
3. **Encrypt `FileKey` with `Receiver.PublicKey` (embed Version to prevent rollback attacks)**

```plaintext
EncryptedKeyForUserX = PKEEnc(ReceiverPublicKey, (FileKey || FileMetadata.Version))
```

4. **Generate `Invitation` structure**

```go
Invitation {
    FilePointer: FileUUID,
    EncryptedKeyForUserX: EncryptedKeyForUserX,
    SenderSignature: Sign(SenderPrivateKey, FileUUID || EncryptedKeyForUserX)
}
```

5. **Store `Invitation` in `Datastore`**
6. **Return `InvitationUUID` to Sender, and send to Receiver (via external channel)**

Data Storage Changes
New values in `Datastore`:
- `InvitationUUID` (unique identifier)
- `Invitation` structure (Include`FilePointer`,`EncryptedKey`,`SenderSignature`)

2️⃣ `AcceptInvitation(receiver, invitationUUID)`
Goal: Receiver accepts the invitation and stores `FileKey` in `UserMetadata`.

Steps
1. **Fetch `Invitation` from `Datastore`**
2. **Verify `SenderSignature`**

```plaintext
Verify(SenderPublicKey, FileUUID || EncryptedKey, SenderSignature)
```

If signature verification fails, deny access.

3. **Decrypt `FileKey` (and validate `Version`)**

```plaintext
```

````
(FileKey, DecryptedVersion) = PKEDec(ReceiverPrivateKey, EncryptedKey)
if DecryptedVersion < FileMetadata.Version:
    return Error("Old version detected, access denied.")
```

`Version` check prevents attackers from using an outdated `EncryptedKey` to gain access.

4. **Store `FileUUID` and `EncryptedFileKey` to `UserMetadata`**
5. **Update `File.SharedWith[Receiver] = EncryptedKey`**
6. **Store `Updated FileMetadata` to `Datastore`**


### **Data Storage Changes**

In `Datastore`:
- Update `FileMetadata.SharedWith` (add `Receiver`)
- Update `F`ileMetadata.Version` to ensure old keys immediately become invalid


## **3️⃣ Non-Owner Sharing (`NonOwner Sharing`)**
**Goal:** CodaBot (non-owner) shares a file with PintoBot.

### **Steps**
1. CodaBot retrieves `EncryptedFileKey` from `SharedWith`
2. Decrypt `FileKey` (and validate `Version`)

```plaintext
(FileKey, DecryptedVersion) = PKEDec(CodaBotPrivateKey, EncryptedFileKeyForCoda)
if DecryptedVersion < FileMetadata.Version:
    return Error("Old version detected, access denied.")
```

`Version` check prevents `CodaBot` from forging or using outdated keys.

3. **Encrypt `FileKey` with `PintoBotPublicKey` (embed `Version`)**

```plaintext
EncryptedKeyForPinto = PKEEnc(PintoBotPublicKey, (FileKey || FileMetadata.Version))
```

4. **Store the new `Invitation`**
5. **PintoBot accepts `Invitation`, repeating `AcceptInvitation()` logic**
6. **Store `File.SharedWith[PintoBot] = EncryptedKeyForPinto`**
7. **Update CodaBot's child list `File.SharedWith[CodaBot][children].append(Pinto)`**


### **Data Storage Changes**
In Datastore:
- Create new `Invitation`
- Update `FileMetadata.SharedWith[PintoBot]`
````
````

4️⃣ `ChunkKey` Rotation Mechanism
Goal: Even if attackers steal the `ChunkKey`, they can only access data for a short period.

1️⃣ Check if `ChunkKey` has expired
- Check `KeyExpirationTime` to determine if it has expired.

```go
If CurrentTime >= FileChunk.KeyExpirationTime:
    RotateChunkKey()
```

2️⃣ Update `FileMetadata.Version`
Increment `Version` first, ensuring subsequent keys embed the correct `Version` information.

```plaintext
FileMetadata.Version++
```

3️⃣ Generate new `ChunkKey` and encrypt data
- **Generate `NewChunkKey`**
- **Encrypt `ChunkData` using `NewChunkKey`**
- **Encrypt `NewChunkKey` using `NewFileKey`** (and embed `Version`)

```plaintext
NewChunkKey = AES-GenerateKey()
NewEncryptedData = AES-CTR(NewChunkKey, PlaintextChunk)
NewEncryptedChunkKey = AES-Encrypt(NewFileKey, (NewChunkKey || FileMetadata.Version))
```

4️⃣ Update `FileMetadata`
- Update `EncryptedChunkKeys`
- Store `Updated FileMetadata`

```go
fileMetadata.EncryptedChunkKeys[chunkUUID] = NewEncryptedChunkKey
SaveFileMetadata(fileMetadata)
```

Answer to Core Design Questions

1. New Values in Datastore when `CreateInvitation()` is Called

`InvitationUUID` (unique identifier)
`Invitation` (includes `FilePointer`, `EncryptedKeyForUserX`, `SenderSignature`)

2. **New Values in Datastore when `AcceptInvitation()` is Called
`FileMetadata.SharedWith` (Add new users)
`FileMetadata.Version` (Increase `Version` to prevent rollback attacks)
`UserMetadata` (Store `FileUUID` and `EncryptedFileKey`)

3. Difference Between Sharing by Non-Owner and Owner
When shared by a non-owner, `EncryptedFileKey` also embeds `Version` to prevent rollback attacks.
The structure of the `Invitation` remains consistent; the sharing process by non-owners is fully compatible with owner sharing.

4. Security of ChunkKey Rotation
`ChunkKey` rotation only re-encrypts expired `ChunkData`, avoiding performance degradation.
Each `EncryptedChunkKey` includes a `Version` to prevent revoked users from restoring access via old `ChunkKeys`.

RevokeAccess Steps

1️⃣ Delete `File.SharedWith[targetUser]` and Recursively Remove Sub-Users
Goal: Use an inheritance chain to find all sub-users of `targetUser` and remove their access rights.

- Delete `File.SharedWith[targetUser]`
- Use `targetUser`'s `Children` list to recursively delete its sub-users (`D`, `E`, `F`)

🔹 `SharedWith` Structure (with Inheritance Chain)

```
SharedWith = {
   "B": { EncryptedFileKeyForB: ..., Children: ["D", "E", "F"] },
   "C": { EncryptedFileKeyForC: ..., Children: ["G"] }
}
```

🔹 Example Code

```go
delete(File.SharedWith, "B")
For each child in File.SharedWith["B"].Children:
   delete(File.SharedWith, child)
```

`B`, `D`, `E`, `F` all lose access rights
Only need to revoke `B`; sub-users are automatically invalidated, reducing manual operations

2️⃣ Generate New `FileKey`
- Generate a new key to ensure that the old `FileKey` for `B`, `D`, `E`, `F` cannot decrypt the data.

```plaintext
NewFileKey = AES-CTR-GenerateKey()
```

Even if the old `FileKey` is recorded by an attacker, it cannot decrypt new `ChunkKeys`.

3️⃣ Re-encrypt `ChunkKeys` with the New `FileKey`
- Re-encrypt `ChunkKeys` with the new `FileKey` to ensure revoked users' old keys are invalid.

```plaintext
For each ChunkKey:
    NewEncryptedChunkKey = AES-Encrypt(NewFileKey, ChunkKey)
```

Even if `B` records the old `ChunkKey`, their old key will still be unable to decrypt it.

4️⃣ Increment `FileMetadata.Version`
- After each revocation, `Version++` is used to ensure that the revoked user's `EncryptedFileKeyForUserX` is tied to the new version.

```plaintext
FileMetadata.Version++
```

Even if a revoked user saves the old `EncryptedFileKeyForUserX`, it cannot decrypt the data due to the changed `Version`.

5️⃣ Update `EncryptedFileKeyForUserX` for Remaining Users like `C`, `G`
Re-encrypt the `NewFileKey` for remaining authorized users like `C`, `G`, and embed the `Version` information.

```plaintext
For each User in SharedWith:
    SharedWith[User] = PKEEnc(UserPublicKey, (NewFileKey || FileMetadata.Version))
```

Ensure that `C` and `G`'s `EncryptedFileKeyForUserX` are updated to the new version, so they do not need to re-accept the `Invitation` upon access.

6️⃣ Calculate New `HMAC(FileMetadata)`
- Recalculate the HMAC of `FileMetadata` to ensure integrity.

```plaintext
NewHMAC = HMAC_SHA256(NewFileKey, UpdatedFileMetadata)
```

Even if an attacker maliciously replaces `FileMetadata` with an old version, the `HMAC` check will fail and trigger access denial.

7️⃣ Execute `DummyChunk` Random Write
- Generate a fake `DummyChunk` to masquerade as real update data, preventing a `Revoked User Adversary` from detecting update frequencies through `Datastore`.

```plaintext
If Random(0, 1) == 1:
    Generate DummyChunk
    Store DummyChunk to Datastore
```

Attackers cannot detect the actual update times by observing the `Datastore`, further enhancing privacy.

Answering Core Design Questions

1. What values need to be updated during revocation?

`FileMetadata.Version`
`FileMetadata.HMAC`
`FileMetadata.EncryptedChunkKeys` (encrypted with `NewFileKey`)
`File.SharedWith` list (remove `B`, `D`, `E`, `F`; update `C`, `G`)

2. How to ensure `C` and `G` can still access?

After re-encrypting `NewFileKey` with `PKEEnc()`, assign it to `C`, `G`
`EncryptedFileKeyForUserX` for `C` and `G` now includes the new `Version`, matching `FileMetadata.Version`

3. How to ensure `B`, `D`, `E`, `F` lose access?

Automatically revoke `B`'s sub-users using the inheritance chain
Delete `SharedWith[B]`, recursively remove `B`'s sub-users (`D`, `E`, `F`)
Re-encrypt `ChunkKeys` with `NewFileKey`, invalidating old `FileKey`
Increment `Version` to ensure the old `EncryptedFileKeyForUserX` is invalid

4. How to prevent `Revoked User Adversary` from regaining access?

Use the `Revoke List` mechanism to reject revoked users in `AcceptInvitation()`
Bind `Version` to `EncryptedFileKeyForUserX` to ensure old keys are invalid for decryption

5. How to prevent `Revoked User Adversary` from detecting future updates?

Use the `DummyChunk` mechanism to randomly fake writes in `Datastore`, obfuscating real update times
Increment `Version` to force clients to download the updated `FileMetadata` with each version, preventing old cache attacks

Design Doc_修改.md

Download

** 🔹 数据结构设计**

🔑 用户元数据 (UserMetadata)

```go
type UserMetadata struct {
    EncryptedPrivateKey []byte    // 加密存储的用户私钥
    PublicKey []byte              // 用户公钥
    SignatureKey []byte           // 用户签名密钥
    RootFilePointer []byte        // 指向用户文件系统的根
    FileMappings map[string]UUID  // 文件名 → 文件UUID 映射
}
```

✅ **存储用户的加密密钥和文件映射，确保访问安全性**

📁 文件 (File) - 逻辑层

```go
type File struct {
    VersionVector map[string]int  // 版本控制 (CRDT 或并发编辑支持)
    Owner string                  // 文件所有者
    EncryptedFileKey []byte       // `FileKey`，用于解密 `ChunkKeys`
    // 用owner自己的公钥加密(仅供自己用)
    OwnerSignature []byte         // 文件所有者签名，防止篡改
    SharedWith map[string][]byte  // 共享用户列表 (用户 → EncryptedFileKeyForUserx)
    CRDT_Log  []LogEntry          // 维护冲突解析日志
}
```

✅ **管理访问控制 & 共享信息**，不存 `ChunkList`
✅ **`SharedWith` 以支持非所有者共享和级联撤销**

📑 文件元数据 (FileMetadata) - 物理存储层

```go
type FileMetadata struct {
    Version int                   // 版本号，防止回滚攻击
    ChunkList []UUID              // UUID 列表，每个 chunk 对应一个 FileChunk
    EncryptedChunkKeys map[string][]byte  // 每个Chunk的密钥都可能不同 (UUID → EncryptedChunkKey)
    OwnerSignature []byte         // 文件所有者签名，防止篡改
    HMAC []byte                   // 元数据完整性保护
}
```

✅ **`EncryptedChunkKeys` 从数组变为映射 `map[string][]byte`，以支持 `ChunkKey` 轮换机制。**
✅ **每个 `Chunk` 使用独立的 `ChunkKey`，满足 `ChunkKey` 过期机制。**

📦 文件数据块 (FileChunk) - 数据存储层

```go
type FileChunk struct {
    ChunkUUID string            // 唯一标识
    EncryptedData []byte        // AES-CTR 加密后的 Chunk 数据
    EncryptedChunkKey []byte    // 用 `FileKey` 加密的 `ChunkKey`
    HMACTag []byte              // 完整性校验
    KeyExpirationTime int64     // 过期时间戳（Unix Time）
    ChunkNumber int             // 顺序编号
    DeviceID string             // 设备来源
    PreviousChunkHash []byte    // 可选字段，用于验证顺序完整性
}
```

✅ **每个 `Chunk` 拥有独立的 `ChunkKey`，并带有 `KeyExpirationTime` 字段**
✅ **即使 `ChunkKey` 泄露，也只能在短时间内访问，超时后需重新加密**

📩 邀请结构 (Invitation) - 共享文件

```go
type Invitation struct {
    FilePointer []byte      // 指向共享文件的 UUID
    EncryptedKeyForUserx []byte     // 用 Receiver 的公钥加密的 `FileKey`
    SenderSignature []byte  // 用 Sender 私钥签名，防止篡改
}
```

✅ **共享时，`EncryptedKeyForUserx` 仅允许 Receiver 访问，防止篡改**
✅ **撤销后，该 `Invitation` 立即失效，Receiver 不能再解密 `FileKey`**

📌 设计方案：Efficient Append

📌 1️⃣ 检查 `FileMetadata.Version` 是否匹配

- 若 `Version` 不匹配，**重新获取 `FileMetadata`**。

```go
if CachedVersion < FileMetadata.Version:
    ForceRefreshMetadata()
```

✅ 确保追加操作基于最新 `FileMetadata`。

📌 2️⃣ 解密 `FileKey`，然后解密 `ChunkKeys`
- 若没有共享，用`ownerPrivateKey`解密`EncryptedFileKey`，确保 `Version` 校验通过。
- 若有共享，用`ReceiverPrivateKey`解密 `EncryptedFileKeyForUserX`，确保 `Version` 校验通过。
- 解密 `EncryptedChunkKeys`，并检查 `Version`。

```go
(FileKey, DecryptedVersion) = PKEDec(ReceiverPrivateKey, EncryptedFileKeyForUserX)
if DecryptedVersion < FileMetadata.Version:
    return Error("Old version detected, access denied.")
```

✅ 版本检查确保旧 `FileKey` 无法访问数据。

📌 3️⃣ 生成新的 `ChunkKey` 并加密数据

- 为追加的内容生成新的 `ChunkKey`
- 使用 `AES-CTR(ChunkKey, append_data)` 加密数据

```plaintext
NewChunkKey = AES-GenerateKey()
EncryptedChunk = AES-CTR(NewChunkKey, append_data)
```

✅ `ChunkKey` 随机生成，保证数据独立性和安全性。

📌 4️⃣ 计算 `HMAC`

- 计算 `HMAC(ChunkData || FileMetadata.Version || ChunkUUID)`

```plaintext
HMAC_Chunk = HMAC_SHA256(NewChunkKey, (EncryptedChunk || FileMetadata.Version || ChunkUUID))
```

✅ `HMAC` 绑定 `Version`，确保旧密钥失效。

📌 5️⃣ 更新 `FileMetadata`

- 追加 `ChunkUUID`
- 重新加密 `ChunkKey` 并存入 `EncryptedChunkKeys`

```plaintext
EncryptedChunkKey = AES-Encrypt(NewFileKey, (NewChunkKey || FileMetadata.Version))
```

```
FileMetadata.ChunkList.append(NewChunkUUID)
FileMetadata.EncryptedChunkKeys[NewChunkUUID] = EncryptedChunkKey
```

✅ 追加操作仅更新元数据和 `ChunkKey`，不会重加密原始数据。

📌 6️⃣ 更新 `FileMetadata.Version` 和 `HMAC`

- `Version++` 确保旧 `ChunkKey` 无法解密新数据。
- 重新计算 `HMAC(FileMetadata)` 以防止伪造。

```plaintext
FileMetadata.Version++
FileMetadata.HMAC = HMAC_SHA256(NewFileKey, UpdatedFileMetadata)
```

✅ `Version` 变更确保所有旧数据失效。

**📌 共享设计方案

📌 1️⃣ `CreateInvitation(sender, receiver, fileUUID)`

📌 **目标**：生成一个共享邀请，并加密 `FileKey` 以安全共享

📍 步骤

1. **获取 `FileUUID`**
2. **用`OwnerPrivateKey`解密`EncryptedFileKey`获得``FileKey**
3. **用 `Receiver.PublicKey` 加密 `FileKey` (嵌入 `Version` 以防回滚攻击)**

```plaintext
EncryptedKeyForUserX = PKE_Encrypt(ReceiverPublicKey, (FileKey || FileMetadata.Version))
```

4. 生成 `Invitation` 结构

```go
Invitation {
    FilePointer: FileUUID,
    EncryptedKeyForUserX: EncryptedKeyForUserX,
    SenderSignature: Sign(SenderPrivateKey, FileUUID || EncryptedKeyForUserX)
}
```

5. 存储 `Invitation` 到 Datastore
6. 返回 `InvitationUUID` 给 `Sender`，并传送给 `Receiver`（通过外部信道）

📍 数据存储变化

✅ 在 Datastore 中新增：

- `InvitationUUID` (作为唯一标识)
- `Invitation` 结构体 (包含 `FilePointer`、`EncryptedKey`、`SenderSignature`)

📌 2️⃣ `AcceptInvitation(receiver, invitationUUID)`

📌 **目标**：Receiver 接受邀请，并将 `FileKey` 存入 `UserMetadata`

📍 步骤

1. **从 Datastore 获取 `Invitation`**
2. **验证 `SenderSignature`**

```plaintext
Verify(SenderPublicKey, FileUUID || EncryptedKey, SenderSignature)
```

✅ 如果签名验证失败，拒绝访问。

3. **解密 `FileKey` (并校验 `Version`)**

```plaintext
(FileKey, DecryptedVersion) = PKE_Decrypt(ReceiverPrivateKey, EncryptedKey)
if DecryptedVersion < FileMetadata.Version:
    return Error("Old version detected, access denied.")
```

✅ `Version` 检查防止攻击者使用过期 `EncryptedKey` 获取访问权限。

4. **存储 `FileUUID` 和 `EncryptedFileKey` 到 `UserMetadata`**
5. **更新 `File.SharedWith[Receiver] = EncryptedKey`**
6. **存储 `Updated FileMetadata` 到 Datastore**

📍 数据存储变化

✅ 在 Datastore 中：

- 更新 `FileMetadata` 的 `SharedWith` (新增 `Receiver`)
- 更新 `FileMetadata.Version`，确保旧密钥立即失效

📌 3️⃣ 非所有者共享 (`NonOwner Sharing`)

📌 **目标**：CodaBot（非所有者）共享文件给 PintoBot

📍 步骤

1. **CodaBot 从 `SharedWith` 获取 `EncryptedFileKey`**
2. **解密 `FileKey` (并校验 `Version`)**

```plaintext
(FileKey, DecryptedVersion) = PKE_Decrypt(CodaBotPrivateKey, EncryptedFileKeyForCoda)
if DecryptedVersion < FileMetadata.Version:
    return Error("Old version detected, access denied.")
```

✅ `Version` 检查防止 `CodaBot` 伪造或使用旧密钥。

3. **用 `PintoBotPublicKey` 加密 `FileKey` (嵌入 `Version`)**

```plaintext
EncryptedKeyForPinto = PKE_Encrypt(PintoBotPublicKey, (FileKey || FileMetadata.Version))
```

4. **存储新的 `Invitation`**
5. **PintoBot 接受 `Invitation`，重复 `AcceptInvitation()` 逻辑**
6. **存储 `File.SharedWith[PintoBot] = EncryptedKeyForPinto`**
7. **更新CodaBot的child列表`File.SharedWith[CodaBot][children].append(Pinto)`**

📍 数据存储变化

✅ 在 Datastore 中：

- 创建新的 `Invitation`
- 在 `FileMetadata` 中更新 `SharedWith[PintoBot]`

📌 4️⃣ `ChunkKey` 轮换机制

📌 **目标**：即使攻击者窃取 `ChunkKey`，也只能在短时间内访问。

📌 1️⃣ 检查 `ChunkKey` 是否过期

- 检查 `KeyExpirationTime`，判断是否过期。

```go
If CurrentTime >= FileChunk.KeyExpirationTime:
   RotateChunkKey()
```

📌 2️⃣ 更新 `FileMetadata.Version`

✅ **先递增 `Version`，确保之后生成的密钥嵌入正确 `Version` 信息。**

```plaintext
FileMetadata.Version++
```

📌 3️⃣ 生成新的 `ChunkKey` 并加密数据

- **生成 `NewChunkKey`**
- **用 `NewChunkKey` 加密 `ChunkData`**
- **用 `NewFileKey` 加密 `NewChunkKey`** (并嵌入 `Version`)

```plaintext
NewChunkKey = AES-GenerateKey()
NewEncryptedData = AES-CTR(NewChunkKey, PlaintextChunk)
NewEncryptedChunkKey = AES-Encrypt(NewFileKey, (NewChunkKey || FileMetadata.Version))
```

📌 4️⃣ 更新 `FileMetadata`

- 更新 `EncryptedChunkKeys`
- 存储 `Updated FileMetadata`

```go
fileMetadata.EncryptedChunkKeys[chunkUUID] = NewEncryptedChunkKey
SaveFileMetadata(fileMetadata)
```

🔎 回答 Design Question 核心问题

📌 1. `CreateInvitation()` 时，Datastore 中新增的值

✅ `InvitationUUID` (唯一标识)
✅ `Invitation` (包含 `FilePointer`、`EncryptedKeyForUserX`、`SenderSignature`)

📌 2. `AcceptInvitation()` 时，Datastore 中更新的值

✅ `FileMetadata.SharedWith` (添加新用户)
✅ `FileMetadata.Version` (增加 `Version`，防止回滚攻击)
✅ `UserMetadata` (存储 `FileUUID` 和 `EncryptedFileKey`)

📌 3. `NonOwner Sharing` 时，与所有者共享的区别

✅ 非所有者共享时，`EncryptedFileKey` 同样嵌入 `Version`，避免回滚攻击。
✅ `Invitation` 结构保持一致，非所有者共享流程与所有者共享完全兼容。

📌 4. 轮换 `ChunkKey` 时的安全性

✅ `ChunkKey` 轮换机制仅重新加密过期 `ChunkData`，避免性能下降。
✅ 每个 `EncryptedChunkKey` 包含 `Version`，防止已撤销用户通过旧 `ChunkKey` 恢复访问。

📍 RevokeAccess 步骤

1️⃣ 删除 `File.SharedWith[targetUser]` 并递归删除子用户

目标：通过继承链机制，找到 `targetUser` 的所有子用户，并移除它们的访问权限。

- 删除 `File.SharedWith[targetUser]`
- 使用 `targetUser` 的 `Children` 列表，递归删除其子用户 (`D`, `E`, `F`)

🔹 `SharedWith` 结构 (加入继承链)

```
SharedWith = {
   "B": { EncryptedFileKeyForB: ..., Children: ["D", "E", "F"] },
   "C": { EncryptedFileKeyForC: ..., Children: ["G"] }
}
```

🔹 示例代码

```go
delete(File.SharedWith, "B")
For each child in File.SharedWith["B"].Children:
   delete(File.SharedWith, child)
```

```

✅ **`B`, `D`, `E`, `F` 均失去访问权限**
✅ **只需撤销 `B`，子用户自动递归失效，减少手动操作**

---

### **2️⃣ 生成新的 `FileKey`**

- 生成新密钥，确保 `B`, `D`, `E`, `F` 的旧 `FileKey` 无法解密数据。

```plaintext
NewFileKey = AES-CTR-GenerateKey()
```

✅ **旧 `FileKey` 即使被攻击者记录，也无法解密新 `ChunkKeys`**

3️⃣ 使用 `NewFileKey` 重新加密 `ChunkKeys`

- 使用 `NewFileKey` 重新加密 `ChunkKeys`，确保已撤销用户的旧密钥无效。

```plaintext
For each ChunkKey:
    NewEncryptedChunkKey = AES-Encrypt(NewFileKey, ChunkKey)
```

✅ **即使 `B` 记录下旧 `ChunkKey`，其旧密钥仍无法解密。**

4️⃣ 增加 `FileMetadata.Version`

- 每次撤销后，`Version++`，确保撤销后的 `EncryptedFileKeyForUserX` 与 `Version` 绑定。

```plaintext
FileMetadata.Version++
```

✅ **已撤销用户即使保存旧的 `EncryptedFileKeyForUserX`，因 `Version` 变更而无法解密数据。**

5️⃣ 为 `C`、`G` 等剩余用户更新 `EncryptedFileKeyForUserX`

✅ 为 `C`、`G` 等剩余授权用户重新加密 `NewFileKey`，并嵌入 `Version` 信息。

```plaintext
For each User in SharedWith:
    SharedWith[User] = PKE_Encrypt(UserPublicKey, (NewFileKey || FileMetadata.Version))
```
```

✅ **确保 `C` 和 `G` 的 `EncryptedFileKeyForUserX` 也更新为新版本，访问时无需重新接受 `Invitation`**

6️⃣ 计算新的 `HMAC(FileMetadata)`

- 重新计算 `FileMetadata` 的 HMAC，确保其完整性。

```plaintext
NewHMAC = HMAC_SHA256(NewFileKey, UpdatedFileMetadata)
```

✅ **即使攻击者恶意替换 `FileMetadata` 为旧版本，也会因 `HMAC` 校验失败而触发拒绝访问。**

7️⃣ 执行 `DummyChunk` 随机写入

- 生成虚假的 `DummyChunk`，伪装成真实更新数据，防止 `Revoked User Adversary` 通过监测 `Datastore` 获取更新频率。

```plaintext
If Random(0, 1) == 1:
    Generate DummyChunk
    Store DummyChunk to Datastore
```

✅ **攻击者无法通过观察 `Datastore` 检测何时更新，进一步提升隐私。**

🔎 回答 Design Question 的核心问题

📌 1. 撤销时需要更新哪些值？

✅ `FileMetadata.Version`
✅ `FileMetadata.HMAC`
✅ `FileMetadata.EncryptedChunkKeys` (使用 `NewFileKey` 加密)
✅ `File.SharedWith` 列表 (删除 `B`、`D`、`E`、`F`；更新 `C`、`G`)

📌 2. 如何确保 `C` 和 `G` 仍能访问？

✅ `NewFileKey` 使用 `PKE_Encrypt()` 重新加密后，分配给 `C`、`G`
✅ `C` 和 `G` 的 `EncryptedFileKeyForUserX` 中嵌入新 `Version`，匹配 `FileMetadata.Version`

📌 3. 如何确保 `B`, `D`, `E`, `F` 均失去访问权限？

✅ 使用继承链 (`Hierarchy Chain`) 自动撤销 `B` 的子用户
✅ 删除 `SharedWith[B]`，递归移除 `B` 的下级 (`D`, `E`, `F`)
✅ 使用 `NewFileKey` 重新加密 `ChunkKeys`，旧 `FileKey` 失效
✅ `Version++` 确保旧 `EncryptedFileKeyForUserX` 无效

📌 4. 如何防止 `Revoked User Adversary` 重新获得访问权限？

✅ `Revoke List` 机制：已撤销用户在 `AcceptInvitation()` 时被拒绝
✅ `Version` 绑定 `EncryptedFileKeyForUserX`，旧密钥解密时自动失效

📌 5. 如何防止 `Revoked User Adversary` 探测未来更新？

✅ 使用 `DummyChunk` 机制，在 `Datastore` 中随机伪造写入行为，混淆真实更新时机
✅ `Version` 递增，确保每次 `FileMetadata` 更新时客户端强制重新下载，防止旧缓存攻击

README.md

Download

Project 2 Starter Code

This repository contains the starter code for Project 2!

For comprehensive documentation, see the Project 2 Spec (https://cs161.org/proj2/).

A friendly request: please do not make your solution public!

Write your implementation in `client/client.go` and your integration tests in `client_test/client_test.go`. Optionally, you can also use `client/client_unittest.go` to write unit tests (e.g: to test your helper functions).

To test your implementation, run `go test -v` inside of the `client_test` directory. This will run all tests in both `client/client_unittest.go` and `client_test/client_test.go`.

Project Members

Fill in this section with the student IDs of all the members in your project group.

Partner 1 Name: Shuhua Zhang

Partner 1 SID: 3039657298

Partner 1 Email: sh.zhang@berkeley.edu

Partner 2 Name (if applicable): Ruihuan Gao

Partner 2 SID (if applicable): 3040857641

Partner 2 Email (if applicable): aimee0704@berkeley.edu

Also add a link to this repo below (should start with https://github.com/cs161-students/).

Link to this Github repo:https://github.com/cs161-students/sp25-proj2-aasas.git

client/client.go

Download

```go
package client

// CS 161 Project 2

// Only the following imports are allowed! ANY additional imports
// may break the autograder!
// - bytes
// - encoding/hex
// - encoding/json
// - errors
// - fmt
// - github.com/cs161-staff/project2-userlib
// - github.com/google/uuid
// - strconv
// - strings

import (
	"encoding/json"

	userlib "github.com/cs161-staff/project2-userlib"
	"github.com/google/uuid"

	// hex.EncodeToString(...) is useful for converting []byte to string

	// Useful for string manipulation

	// Useful for formatting strings (e.g. `fmt.Sprintf`).
	"fmt"

	// Useful for creating new error messages to return using errors.New("...")
	"errors"
	// "bytes"
	// "crypto/rsa"
	// "crypto/x509"
	// "encoding/json"
	// "encoding/pem"

	// Optional.
	_ "strconv"
)

// This serves two purposes: it shows you a few useful primitives,
// and suppresses warnings for imports not being used. It can be
// safely deleted!
func someUsefulThings() {

	// Creates a random UUID.
	randomUUID := uuid.New()

```

```
50      // Prints the UUID as a string. %v prints the value in a default format.
51      // See https://pkg.go.dev/fmt#hdr-Printing for all Golang format string flags.
52      userlib.DebugMsg("Random UUID: %v", randomUUID.String())
53
54      // Creates a UUID deterministically, from a sequence of bytes.
55      hash := userlib.Hash([]byte("user-structs/alice"))
56      deterministicUUID, err := uuid.FromBytes(hash[:16])
57      if err != nil {
58          // Normally, we would `return err` here. But, since this function doesn't return anything,
59          // we can just panic to terminate execution. ALWAYS, ALWAYS, ALWAYS check for errors! Your
60          // code should have hundreds of "if err != nil { return err }" statements by the end of this
61          // project. You probably want to avoid using panic statements in your own code.
62          panic(errors.New("An error occurred while generating a UUID: " + err.Error()))
63      }
64      userlib.DebugMsg("Deterministic UUID: %v", deterministicUUID.String())
65
66      // Declares a Course struct type, creates an instance of it, and marshals it into JSON.
67      type Course struct {
68          name      string
69          professor []byte
70      }
71
72      course := Course{"CS 161", []byte("Nicholas Weaver")}
73      courseBytes, err := json.Marshal(course)
74      if err != nil {
75          panic(err)
76      }
77
78      userlib.DebugMsg("Struct: %v", course)
79      userlib.DebugMsg("JSON Data: %v", courseBytes)
80
81      // Generate a random private/public keypair.
82      // The "_" indicates that we don't check for the error case here.
83      var pk userlib.PKEEncKey
84      var sk userlib.PKEDecKey
85      pk, sk, _ = userlib.PKEKeyGen()
86      userlib.DebugMsg("PKE Key Pair: (%v, %v)", pk, sk)
87
88      // Here's an example of how to use HBKDF to generate a new key from an input key.
89      // Tip: generate a new key everywhere you possibly can! It's easier to generate new keys on the fly
90      // instead of trying to think about all of the ways a key reuse attack could be performed. It's also
    easier to
91      // store one key and derive multiple keys from that one key, rather than
92      originalKey := userlib.RandomBytes(16)
93      derivedKey, err := userlib.HashKDF(originalKey, []byte("mac-key"))
94      if err != nil {
95          panic(err)
96      }
97      userlib.DebugMsg("Original Key: %v", originalKey)
98      userlib.DebugMsg("Derived Key: %v", derivedKey)
99
100     // A couple of tips on converting between string and []byte:
```

```
101         // To convert from string to []byte, use []byte("some-string-here")
102         // To convert from []byte to string for debugging, use fmt.Sprintf("hello world: %s", some_byte_arr).
103         // To convert from []byte to string for use in a hashmap, use hex.EncodeToString(some_byte_arr).
104         // When frequently converting between []byte and string, just marshal and unmarshal the data.
105         //
106         // Read more: https://go.dev/blog/strings
107
108         // Here's an example of string interpolation!
109         _ = fmt.Sprintf("%s_%d", "file", 1)
110 }
111
112 // This is the type definition for the User struct.
113 // A Go struct is like a Python or Java class - it can have attributes
114 // (e.g. like the Username attribute) and methods (e.g. like the StoreFile method below).
115 // type User struct {
116 //     Username string
117
118 //     // You can add other attributes here if you want! But note that in order for attributes to
119 //     // be included when this struct is serialized to/from JSON, they must be capitalized.
120 //     // On the flipside, if you have an attribute that you want to be able to access from
121 //     // this struct's methods, but you DON'T want that value to be included in the serialized value
122 //     // of this struct that's stored in datastore, then you can use a "private" variable (e.g. one that
123 //     // begins with a lowercase letter).
124 // }
125
126 type User struct {
127         Username   string
128         MasterKey  []byte
129         FileKey    []byte
130         PublicKey  userlib.PKEEncKey
131         PrivateKey userlib.PKEDecKey
132         SignKey    userlib.DSSignKey
133         VerifyKey  userlib.DSVerifyKey
134 }
135
136 type FileView struct {
137         MetadataUUID uuid.UUID
138         EncKey       []byte
139         HMACKey      []byte
140         Status       string           // Own | Shared | Received
141         PendingInv   map[string]uuid.UUID // recipientUsername → invitationPtr
142 }
143
144 type FileMetadata struct {
145         Owner        string
146         FileName     string
147         HeadPtr      uuid.UUID
148         TailPtr      uuid.UUID
149         NumberChunk  int
150         FileEncKey   []byte
151         HMACKey      []byte
152         ShareListAddr uuid.UUID
```

```
153         Version      uint64
154 }
155
156 type FileChunk struct {
157         Data []byte    // 加密后的文件内容（或明文，然后用 FileEncKey 加密）
158         Next uuid.UUID // 指向下一个 Chunk（或空 UUID 表示终止）
159 }
160
161 type ShareEntry struct {
162         Sender      string
163         Recipient   string
164         FileKey    []byte // fileKey of recipient
165         MetadataUUID uuid.UUID
166         Filename    string // recipient's filename (may be different from sender's filename)
167 }
168
169 // 文件共享邀请结构
170 type Invitation struct {
171         EncView     []byte // AES 加密的 FileView
172         EncryptedKey []byte // 用 RSA 加密的对称密钥
173         SenderSig    []byte
174 }
175
176 type SignedShareList struct {
177         List map[string][]ShareEntry
178 }
179
180 // helper function
181
182 // DeriveKeys generates encryption and MAC keys from a key.
183 func DeriveKeys(pdk []byte, encInput []byte, hmacInput []byte) (encKey []byte, macKey []byte, err error)
    {
184         encKey, err = userlib.HashKDF(pdk, encInput)
185         if err != nil {
186                 return nil, nil, err
187         }
188         encKey = encKey[:16]
189
190         macKey, err = userlib.HashKDF(pdk, hmacInput)
191         if err != nil {
192                 return nil, nil, err
193         }
194         macKey = macKey[:16]
195         return encKey, macKey, nil
196 }
197
198 // AuthEncrypt encrypts and generates HMAC for integrity.
199 func AuthEncrypt(key []byte, plaintext []byte) (ciphertext []byte, hmacTag []byte, err error) {
200         encKey, macKey, err := DeriveKeys(key, []byte("Encryption"), []byte("HMAC"))
201         if err != nil {
202                 return nil, nil, err
203         }
```

```
204         iv := userlib.RandomBytes(userlib.AESBlockSizeBytes)
205         ciphertext = userlib.SymEnc(encKey, iv, plaintext)
206         hmacTag, err = userlib.HMACEval(macKey, ciphertext)
207         return ciphertext, hmacTag, err
208 }
209
210 // AuthDecrypt validates HMAC and decrypts.
211 func AuthDecrypt(key []byte, ciphertext []byte, hmacTag []byte) (plaintext []byte, err error) {
212         encKey, macKey, err := DeriveKeys(key, []byte("Encryption"), []byte("HMAC"))
213         if err != nil {
214                 return nil, err
215         }
216         expectedTag, err := userlib.HMACEval(macKey, ciphertext)
217         if err != nil || !userlib.HMACEqual(hmacTag, expectedTag) {
218                 return nil, errors.New("HMAC verification failed")
219         }
220         plaintext = userlib.SymDec(encKey, ciphertext)
221         return plaintext, nil
222 }
223
224 func EasyEncrypt(encKey []byte, macKey []byte, plaintext []byte) (ciphertext []byte, hmacTag []byte, err
    error) {
225         iv := userlib.RandomBytes(userlib.AESBlockSizeBytes)
226         ciphertext = userlib.SymEnc(encKey, iv, plaintext)
227         hmacTag, err = userlib.HMACEval(macKey, ciphertext)
228         return ciphertext, hmacTag, err
229 }
230
231 func EasyDecrypt(encKey []byte, macKey []byte, ciphertext []byte, hmacTag []byte) (plaintext []byte, err
    error) {
232         expectedTag, err := userlib.HMACEval(macKey, ciphertext)
233         if err != nil || !userlib.HMACEqual(hmacTag, expectedTag) {
234                 userlib.DebugMsg("HMAC verification failed")
235                 return nil, errors.New("HMAC verification failed")
236         }
237         plaintext = userlib.SymDec(encKey, ciphertext)
238         return plaintext, nil
239 }
240
241 // ZeroBytes 用 0 覆盖字节切片，防止敏感数据仍在内存中
242 func ZeroBytes(data []byte) {
243         for i := range data {
244                 data[i] = 0
245         }
246 }
247
248 // HybridEncrypt 使用混合加密方案加密数据，返回加密后的对称密钥和加密数据
249 func HybridEncrypt(publicKey userlib.PKEEncKey, plaintext []byte) (encryptedSymKey []byte,
    encryptedData []byte, err error) {
250         // 生成随机的对称密钥（AES-128）
251         symKey := userlib.RandomBytes(userlib.AESKeySizeBytes)
252         iv := userlib.RandomBytes(userlib.AESBlockSizeBytes)
```

```
253         encryptedData = userlib.SymEnc(symKey, iv, plaintext)
254         encryptedSymKey, err = userlib.PKEEnc(publicKey, symKey)
255         if err != nil {
256             return nil, nil, fmt.Errorf("公钥加密失败: %v", err)
257         }
258         return encryptedSymKey, encryptedData, nil
259 }
260
261 // HybridDecrypt 使用混合加密方案解密数据，返回原始明文
262 func HybridDecrypt(privateKey userlib.PKEDecKey, encryptedSymKey []byte, encryptedData []byte)
    (symKey []byte, plaintext []byte, err error) {
263         symKey, err = userlib.PKEDec(privateKey, encryptedSymKey)
264         if err != nil {
265             return nil, nil, fmt.Errorf("私钥解密失败: %v", err)
266         }
267         if len(symKey) != userlib.AESKeySizeBytes {
268             return nil, nil, errors.New("解密得到的对称密钥长度不合法")
269         }
270         if len(encryptedData) < userlib.AESBlockSizeBytes {
271             return nil, nil, errors.New("加密数据长度异常")
272         }
273         plaintext = userlib.SymDec(symKey, encryptedData)
274         return symKey, plaintext, nil
275 }
276
277 // File Chunk Helper
278 func SaveFileChunk(fileEncKey []byte, fileHMACKey []byte, id uuid.UUID, chunk *FileChunk) error {
279         chunkBytes, err := json.Marshal(chunk)
280         if err != nil {
281             return errors.New("chunk data fail to encode")
282         }
283
284         iv := userlib.RandomBytes(userlib.AESBlockSizeBytes)
285         chunkEnc := userlib.SymEnc(fileEncKey, iv, chunkBytes)
286
287         // HMAC时加上当前ID
288         hmacInput := append([]byte{}, chunkEnc...)
289         hmacInput = append(hmacInput, id[:]...)
290         chunkHMAC, err := userlib.HMACEval(fileHMACKey, hmacInput)
291
292         if err != nil {
293             return err
294         }
295         userlib.DatastoreSet(id, append(chunkEnc, chunkHMAC...))
296         return nil
297 }
298
299 func LoadFileChunk(fileEncKey []byte, fileHMACKey []byte, id uuid.UUID) (*FileChunk, error) {
300         raw, ok := userlib.DatastoreGet(id)
301         if !ok {
302             return nil, errors.New("file Chunk not exist")
303         }
```

```
304         hmacSize := userlib.HashSizeBytes
305         if len(raw) < 16+hmacSize { // UUID + HMAC
306             return nil, errors.New("chunk data corrupted")
307         }
308
309         chunkHMAC := raw[len(raw)-hmacSize:]
310         chunkEnc := raw[:len(raw)-hmacSize]
311
312         var fileChunk FileChunk
313         hmacInput := append([]byte{}, chunkEnc...)
314         hmacInput = append(hmacInput, id[:]...)
315
316         expectedTag, err := userlib.HMACEval(fileHMACKey, hmacInput)
317         if err != nil || !userlib.HMACEqual(chunkHMAC, expectedTag) {
318             userlib.DebugMsg("HMAC verification failed")
319             return nil, errors.New("HMAC verification failed")
320         }
321         fileChunkByte := userlib.SymDec(fileEncKey, chunkEnc)
322
323         err = json.Unmarshal(fileChunkByte, &fileChunk)
324         if err != nil {
325             return nil, errors.New("chunk data fail to decode")
326         }
327
328         return &fileChunk, nil
329 }
330
331 // File Metadata Helper
332 func SaveFileMetadata(id uuid.UUID, meta *FileMetadata, encKey, hmacKey []byte) error {
333         metaBytes, err := json.Marshal(meta)
334         if err != nil {
335             return err
336         }
337         cipher, tag, err := EasyEncrypt(encKey, hmacKey, metaBytes)
338         if err != nil {
339             return err
340         }
341         userlib.DatastoreSet(id, append(cipher, tag...))
342         return nil
343 }
344
345 func LoadFileMetadata(id uuid.UUID, encKey []byte, HMACKey []byte) (*FileMetadata, error) {
346         raw, ok := userlib.DatastoreGet(id)
347         if !ok {
348             return nil, errors.New("metadata not found")
349         }
350
351         hmacSize := userlib.HashSizeBytes
352         if len(raw) <= hmacSize {
353             return nil, errors.New("invalid metadata length")
354         }
355
```

```
356         metadataHMAC := raw[len(raw)-hmacSize:]
357         metadataEnc := raw[:len(raw)-hmacSize]
358 
359         // Verify HMAC
360         expectedHMAC, err := userlib.HMACEval(HMACKey, metadataEnc)
361         if err != nil {
362             return nil, errors.New("failed to compute HMAC")
363         }
364         if !userlib.HMACEqual(metadataHMAC, expectedHMAC) {
365             return nil, errors.New("metadata HMAC mismatch")
366         }
367 
368         // Decrypt metadata
369         var meta FileMetadata
370         metadataByte := userlib.SymDec(encKey, metadataEnc)
371         err = json.Unmarshal(metadataByte, &meta)
372 
373         if err != nil {
374             return nil, errors.New("fail to decode")
375         }
376         return &meta, nil
377 }
378 
379 func LoadUserFileList(id uuid.UUID, fileListEncKey []byte, fileListHMACKey []byte, isFirst bool) (fileList
    map[string]FileView, err error) {
380         userFileListStore, exist := userlib.DatastoreGet(id)
381         if !exist {
382             if !isFirst {
383                 userlib.DebugMsg("user file list not found")
384                 return nil, errors.New("user file list not found")
385             }
386             return make(map[string]FileView), nil
387         }
388 
389         // Check that the stored data is at least long enough to contain HMAC (64 bytes)
390         if len(userFileListStore) < 64 {
391             userlib.DebugMsg("stored file list too short to contain HMAC")
392             return nil, errors.New("corrupted file list: too short")
393         }
394 
395         // Separate encrypted data and HMAC
396         fileListEnc := userFileListStore[:len(userFileListStore)-64]
397         fileListHMAC := userFileListStore[len(userFileListStore)-64:]
398 
399         // Attempt to decrypt and verify
400         userFileListByte, err := EasyDecrypt(fileListEncKey, fileListHMACKey, fileListEnc, fileListHMAC)
401         if err != nil {
402             userlib.DebugMsg("failed to decrypt user file list")
403             return nil, errors.New("failed to decrypt user file list")
404         }
405 
406         // Attempt to decode JSON
```

```
407         fileList = make(map[string]FileView)
408         err = json.Unmarshal(userFileListByte, &fileList)
409         if err != nil {
410                 userlib.DebugMsg("failed to decode user file list JSON")
411                 return nil, errors.New("failed to decode user file list")
412         }
413
414         return fileList, nil
415 }
416
417 func SaveUserFileList(uuid uuid.UUID, encKey, macKey []byte, list map[string]FileView) error {
418         data, err := json.Marshal(list)
419         if err != nil {
420                 return err
421         }
422         cipher, tag, err := EasyEncrypt(encKey, macKey, data)
423         if err != nil {
424                 return err
425         }
426         userlib.DatastoreSet(uuid, append(cipher, tag...))
427         return nil
428 }
429
430 func LoadSignedShareList(shareListAddr uuid.UUID) (*SignedShareList, error) {
431         shareListBytes, ok := userlib.DatastoreGet(shareListAddr)
432         if !ok {
433                 return nil, errors.New("RevokeAccess: ShareList not found")
434         }
435         var signedList SignedShareList
436         err := json.Unmarshal(shareListBytes, &signedList)
437         if err != nil {
438                 return nil, err
439         }
440         return &signedList, nil
441 }
442
443 func SaveSignedShareList(shareListAddr uuid.UUID, signedList *SignedShareList) error {
444         updatedSignedList, err := json.Marshal(signedList)
445         if err != nil {
446                 return errors.New("error decode ShareList")
447         }
448         userlib.DatastoreSet(shareListAddr, updatedSignedList)
449         return nil
450 }
451
452 func VerifyFileIntegrity(meta *FileMetadata) error {
453         current := meta.HeadPtr
454         count := 0
455         for count < meta.NumberChunk {
456                 chunk, err := LoadFileChunk(meta.FileEncKey, meta.HMACKey, current)
457                 if err != nil {
458                         return errors.New("Verify: fail to load chunk")
```

```
459             }
460             current = chunk.Next
461             count++
462         }
463         if count != meta.NumberChunk {
464             return errors.New("Verify: chunk count mismatch")
465         }
466         return nil
467 }
468
469 // NOTE: The following methods have toy (insecure!) implementations.
470
471 func InitUser(username string, password string) (*User, error) {
472         // 用户名非空
473         if len(username) == 0 {
474             return nil, errors.New("username cannot be empty")
475         }
476
477         // 防重复
478         userUUID, err := uuid.FromBytes(userlib.Hash([]byte("user_" + username))[:16])
479         // userlib.DebugMsg("InitUser: userUUID: %v\n", userUUID)
480         if err != nil {
481             return nil, err
482         }
483         if _, ok := userlib.DatastoreGet(userUUID); ok {
484             return nil, errors.New("InitUser: user already exists")
485         }
486
487         // 密钥对生成
488         publicKey, privateKey, err := userlib.PKEKeyGen()
489         if err != nil {
490             return nil, err
491         }
492         signKey, verifyKey, err := userlib.DSKeyGen()
493         if err != nil {
494             return nil, err
495         }
496
497         // 密码派生密钥
498         salt := userlib.RandomBytes(16)
499         pdk := userlib.Argon2Key([]byte(password), salt, 16)
500
501         //store PKE and DS public keys in keystore
502         userlib.KeystoreSet(username+"_PK", publicKey)
503         userlib.KeystoreSet(username+"_DS", verifyKey)
504
505         // 构造用户数据
506         userdata := User{
507             Username:   username,
508             MasterKey:  pdk,
509             FileKey:    userlib.RandomBytes(16),
510             PublicKey:  publicKey,
```

```
511             PrivateKey: privateKey,
512             SignKey:    signKey,
513             VerifyKey:  verifyKey,
514         }
515
516         //Encrypt user data
517         userDataByte, _ := json.Marshal(userdata)
518         userByteEnc, userHmacTag, err := AuthEncrypt(pdk, userDataByte)
519         if err != nil {
520             return nil, errors.New("InitUser: Fail to encrypt user data")
521         }
522
523         //Store user data
524         finalBytes := append(salt, userByteEnc...)
525         finalBytes = append(finalBytes, userHmacTag...)
526         userlib.DatastoreSet(userUUID, finalBytes)
527
528         // 返回 User
529         return &userdata, nil
530 }
531
532 func GetUser(username string, password string) (*User, error) {
533         userUUID, err := uuid.FromBytes(userlib.Hash([]byte("user_" + username))[:16])
534         if err != nil {
535             return nil, err
536         }
537         stored, ok := userlib.DatastoreGet(userUUID)
538         if !ok {
539             return nil, errors.New("user not found")
540         }
541         if len(stored) < 16+userlib.HashSizeBytes {
542             return nil, errors.New("user corrupted: length too short")
543         }
544
545         // Split salt, userByteEnc, userHmacTag
546         salt := stored[:16]
547         userByteEnc := stored[16 : len(stored)-userlib.HashSizeBytes]
548         userHmacTag := stored[len(stored)-userlib.HashSizeBytes:]
549
550         pdk := userlib.Argon2Key([]byte(password), salt, 16)
551
552         // 验证并解密 user data
553         userdataBytes, err := AuthDecrypt(pdk, userByteEnc, userHmacTag)
554         if err != nil {
555             return nil, errors.New("GetUer: Invalid password or tampered data")
556         }
557
558         var userdata User
559         err = json.Unmarshal(userdataBytes, &userdata)
560         if err != nil {
561             return nil, err
562         }
```

```
563
564         return &userdata, nil
565 }
566
567 func (userdata *User) StoreFile(filename string, data []byte) (err error) {
568         // 生成 FileKey和首个 ChunkKey
569         userFileListID, err := uuid.FromBytes(userlib.Hash([]byte(userdata.Username + "fileList"))[:16])
570         if err != nil {
571                 return err
572         }
573         // userlib.DebugMsg("StoreFile: userFileListID: %v\n", userFileListID)
574         fileListEncKey, fileListHMACKey, err := DeriveKeys(userdata.FileKey, []byte("fileListEncKey"),
    []byte("fileListHMACKey"))
575         if err != nil {
576                 return err
577         }
578
579         fileList, err := LoadUserFileList(userFileListID, fileListEncKey, fileListHMACKey, true)
580         if err != nil {
581                 return err
582         }
583         // userlib.DebugMsg("AFTER STORE Alice fileList: %+v", fileList)
584         fileView, exist := fileList[filename]
585         chunkCount := 1
586
587         // 当前用户的fileList里没有filename, 创建新fileChunk, fileMetadata, fileView
588         if !exist {
589                 // 生成 ChunkUUIDs 和 fileMetadataUUID
590                 chunkUUID := uuid.New()
591                 fileMetadataUUID := uuid.New()
592                 nextchunkUUID := uuid.New()
593                 ShareListAddr := uuid.New()
594                 userlib.DebugMsg("StoreFile: chunkUUID: %v\n", chunkUUID)
595
596                 // Chunk层
597                 // 生成filekeys
598                 fileEncKey, fileHMACKey, err := DeriveKeys(userlib.RandomBytes(16), []byte("fileEncKey"),
    []byte("fileHMACKey"))
599                 if err != nil {
600                         return err
601                 }
602
603                 fileChunk := FileChunk{
604                         Data: data,
605                         Next: nextchunkUUID,
606                 }
607
608                 // 存储 fileChunk
609                 err = SaveFileChunk(fileEncKey, fileHMACKey, chunkUUID, &fileChunk)
610                 if err != nil {
611                         return err
612                 }
```

```
613
614             // metadata层
615             fileMetadata := FileMetadata{
616                 Owner:       userdata.Username,
617                 FileName:    filename,
618                 HeadPtr:     chunkUUID,
619                 TailPtr:     fileChunk.Next,
620                 NumberChunk:  chunkCount,
621                 FileEncKey:   fileEncKey,
622                 HMACKey:     fileHMACKey,
623                 ShareListAddr: ShareListAddr,
624                 Version:     1,
625             }
626             emptyShareList := SignedShareList{
627                 List: make(map[string][]ShareEntry),
628             }
629
630             err = SaveSignedShareList(ShareListAddr, &emptyShareList)
631             if err != nil {
632                 return err
633             }
634
635             metadataEncKey, metadataHMACKey, err := DeriveKeys(userlib.RandomBytes(16),
    []byte("filemetadataEncKey"), []byte("filemetadataHMACKey"))
636             if err != nil {
637                 return err
638             }
639             err = SaveFileMetadata(fileMetadataUUID, &fileMetadata, metadataEncKey,
    metadataHMACKey)
640             if err != nil {
641                 return err
642             }
643
644             //fileView层
645             fileView = FileView{
646                 EncKey:     metadataEncKey,
647                 HMACKey:    metadataHMACKey,
648                 MetadataUUID: fileMetadataUUID,
649                 Status:     "Own",
650             }
651
652             defer ZeroBytes(metadataEncKey)
653             defer ZeroBytes(metadataHMACKey)
654
655             fileList[filename] = fileView
656
657         } else {
658             // 当前用户的fileList里有filename, 直接更新fileChunk和fileMetadata, filekeys和filemetadatakeys
    不变
659             // 获得当前文件的metadata
660             fileMetadataUUID := fileView.MetadataUUID
661             metadataEncKey := fileView.EncKey
```

```
662             metadataHMACKey := fileView.HMACKey
663             fileMetadata, err := LoadFileMetadata(fileMetadataUUID, metadataEncKey,
    metadataHMACKey)
664             if err != nil {
665                     return err
666             }
667
668             fileEncKey := fileMetadata.FileEncKey
669             fileHMACKey := fileMetadata.HMACKey
670
671             //删除旧文件
672             curChunkUUID := fileMetadata.HeadPtr
673             for {
674                     // Load current fileChunk
675                     fileChunk, err := LoadFileChunk(fileEncKey, fileHMACKey, curChunkUUID)
676                     if err != nil {
677                             return errors.New("StoreFile: fail to load old file chunk")
678                     }
679
680                     // delete current fileChunk
681                     userlib.DatastoreDelete(curChunkUUID)
682
683                     if fileChunk.Next == fileMetadata.TailPtr {
684                             break
685                     }
686                     curChunkUUID = fileChunk.Next
687             }
688
689             //Create new file and store
690             // Chunk层
691             // 生成 ChunkUUID
692             newChunkUUID := uuid.New()
693             nextchunkUUID := uuid.New()
694             newFileChunk := FileChunk{
695                     Data: data,
696                     Next: nextchunkUUID,
697             }
698             userlib.DebugMsg("StoreFile: newChunkUUID: %v\n", newChunkUUID)
699
700             // 存储 fileChunk
701             err = SaveFileChunk(fileEncKey, fileHMACKey, newChunkUUID, &newFileChunk)
702             if err != nil {
703                     return err
704             }
705
706             //metadata层
707             fileMetadata.HeadPtr = newChunkUUID
708             fileMetadata.TailPtr = newFileChunk.Next
709             fileMetadata.NumberChunk = chunkCount
710             fileMetadata.Version++
711             err = SaveFileMetadata(fileMetadataUUID, fileMetadata, metadataEncKey, metadataHMACKey)
712             if err != nil {
```

```
713                 return err
714             }
715
716             defer ZeroBytes(metadataEncKey)
717             defer ZeroBytes(metadataHMACKey)
718         }
719
720         // 更新Datastore中用户的fileList
721         err = SaveUserFileList(userFileListID, fileListEncKey, fileListHMACKey, fileList)
722         if err != nil {
723             return err
724         }
725         defer ZeroBytes(fileListEncKey)
726         defer ZeroBytes(fileListHMACKey)
727
728         return nil
729 }
730
731 func (userdata *User) AppendToFile(filename string, data []byte) error {
732         fileListEncKey, fileListHMACKey, err := DeriveKeys(userdata.FileKey, []byte("fileListEncKey"),
    []byte("fileListHMACKey"))
733         if err != nil {
734             return err
735         }
736         userFileListID, err := uuid.FromBytes(userlib.Hash([]byte(userdata.Username + "fileList"))[:16])
737         if err != nil {
738             return err
739         }
740
741         fileList, err := LoadUserFileList(userFileListID, fileListEncKey, fileListHMACKey, false)
742         if err != nil {
743             return err
744         }
745
746         fileView, exist := fileList[filename]
747         if !exist {
748             return errors.New("file view not exist")
749         }
750
751         // 获得当前文件的FileMetadata
752         fileMetadataUUID := fileView.MetadataUUID
753         metadataEncKey := fileView.EncKey
754         metadataHMACKey := fileView.HMACKey
755
756         fileMetadata, err := LoadFileMetadata(fileMetadataUUID, metadataEncKey, metadataHMACKey)
757         if err != nil || fileMetadata == nil {
758             return errors.New("AppendToFile: Failed to load file metadata")
759         }
760         fileEncKey := fileMetadata.FileEncKey
761         fileHMACKey := fileMetadata.HMACKey
762
763         // chunk层
```

```
764         // 生成新的 Chunk
765         newChunkUUID := fileMetadata.TailPtr
766 
767         // 生成新的预留位置用于下一次 Append
768         newNextChunkUUID := uuid.New()
769 
770         // 创建新 Chunk，指向新的预留位置
771         fileChunk := FileChunk{
772             Data: data,
773             Next: newNextChunkUUID, // 更新为新的预留 UUID
774         }
775 
776         // 存储 fileChunk
777         err = SaveFileChunk(fileEncKey, fileHMACKey, newChunkUUID, &fileChunk)
778         if err != nil {
779             return err
780         }
781 
782         // metadata层
783         fileMetadata.TailPtr = newNextChunkUUID
784         fileMetadata.NumberChunk++
785         fileMetadata.Version++
786 
787         // 存储 fileMetadata
788         err = SaveFileMetadata(fileMetadataUUID, fileMetadata, metadataEncKey, metadataHMACKey)
789         if err != nil {
790             return err
791         }
792 
793         defer ZeroBytes(metadataEncKey)
794         defer ZeroBytes(metadataHMACKey)
795         defer ZeroBytes(fileListEncKey)
796         defer ZeroBytes(fileListHMACKey)
797 
798         return nil
799 }
800 
801 func (userdata *User) LoadFile(filename string) ([]byte, error) {
802         fileListEncKey, fileListHMACKey, err := DeriveKeys(userdata.FileKey, []byte("fileListEncKey"),
    []byte("fileListHMACKey"))
803         if err != nil {
804             return nil, err
805         }
806         userFileListID, err := uuid.FromBytes(userlib.Hash([]byte(userdata.Username + "fileList"))[:16])
807         if err != nil {
808             return nil, err
809         }
810 
811         fileList, err := LoadUserFileList(userFileListID, fileListEncKey, fileListHMACKey, false)
812         if err != nil {
813             return nil, err
814         }
```

```
815
816         fileView, exist := fileList[filename]
817         if !exist {
818             return nil, errors.New("file view not exist")
819         }
820
821         // 获得当前文件的FileMetadata
822         fileMetadataUUID := fileView.MetadataUUID
823         metadataEncKey := fileView.EncKey
824         metadataHMACKey := fileView.HMACKey
825         fileMetadata, err := LoadFileMetadata(fileMetadataUUID, metadataEncKey, metadataHMACKey)
826         if err != nil || fileMetadata == nil {
827             return nil, errors.New("LoadFile: Failed to load file metadata:File may have been revoked
    (metadata missing")
828         }
829
830         fileEncKey := fileMetadata.FileEncKey
831         fileHMACKey := fileMetadata.HMACKey
832
833         var fileData []byte
834
835         // Load File Content
836         curChunkUUID := fileMetadata.HeadPtr
837         for {
838             // Load current fileChunk
839             fileChunk, err := LoadFileChunk(fileEncKey, fileHMACKey, curChunkUUID)
840             if err != nil {
841                 return nil, errors.New("LoadFile: chunk data fail to load")
842             }
843
844             fileData = append(fileData, fileChunk.Data...)
845
846             if fileChunk.Next == fileMetadata.TailPtr {
847                 break
848             }
849             curChunkUUID = fileChunk.Next
850         }
851         defer ZeroBytes(metadataEncKey)
852         defer ZeroBytes(metadataHMACKey)
853         defer ZeroBytes(fileListEncKey)
854         defer ZeroBytes(fileListHMACKey)
855         return fileData, nil
856 }
857
858 // CreateInvitation securely creates an invitation to share a file
859 func (userdata *User) CreateInvitation(filename string, recipientUsername string) (invitationPtr
    uuid.UUID, err error) {
860         // 1: Verify recipient's public key
861         recipientPK, ok := userlib.KeystoreGet(recipientUsername + "_PK")
862         if !ok {
863             return uuid.Nil, errors.New("CreateInvitation: recipient public key not found")
864         }
```

```
865
866         // 2: Get FileList
867
868         // Step 1: Derive keys for user's file list
869         fileListEncKey, fileListHMACKey, err := DeriveKeys(userdata.FileKey, []byte("fileListEncKey"),
    []byte("fileListHMACKey"))
870         if err != nil {
871             return uuid.Nil, err
872         }
873         userFileListID, err := uuid.FromBytes(userlib.Hash([]byte(userdata.Username + "fileList"))[:16])
874         if err != nil {
875             return uuid.Nil, err
876         }
877
878         // Step 2: Load the user's file list
879         curFileList, err := LoadUserFileList(userFileListID, fileListEncKey, fileListHMACKey, false)
880         if err != nil {
881             return uuid.Nil, errors.New("CreateInvitation: Failed to load user file list")
882         }
883
884         // Step 3: Check if the fileView exists in the user's list
885         curFv, exist := curFileList[filename]
886         if !exist {
887             return uuid.Nil, errors.New("CreateInvitation: FileView not exist")
888         }
889
890         //3. Check metadata and filenode integrity
891         curFileMetadataUUID := curFv.MetadataUUID
892         curFmEncKey := curFv.EncKey
893         curFmHMAC := curFv.HMACKey
894
895         // Step 4: Load and verify FileMetadata
896         var metadata *FileMetadata
897         metadata, err = LoadFileMetadata(curFileMetadataUUID, curFmEncKey, curFmHMAC)
898         if err != nil {
899             return uuid.Nil, errors.New("CreateInvitation: Failed to load file metadata: " + err.Error())
900         }
901
902         if err := VerifyFileIntegrity(metadata); err != nil {
903             return uuid.Nil, err
904         }
905
906         //4. Generate FileView copy for invitation
907         fvcopy := FileView{
908             MetadataUUID: curFileMetadataUUID,
909             Status:       "Share",
910             HMACKey:      curFmHMAC,
911             EncKey:       curFmEncKey,
912         }
913
914         // Store viewCopy
915         ShareSymKey := userlib.RandomBytes(16)
```

```
916     ShareHMACKey := userlib.RandomBytes(16)
917
918     viewCopyData, err := json.Marshal(fvcopy)
919     if err != nil {
920         return uuid.Nil, errors.New("CreateInvitation: Error marshaling viewCopy")
921     }
922     viewCopyEnc := userlib.SymEnc(ShareSymKey, userlib.RandomBytes(16), viewCopyData)
923     viewCopyHMAC, err := userlib.HMACEval(ShareHMACKey, viewCopyEnc)
924     if err != nil {
925         return uuid.Nil, errors.New("CreateInvitation: Error generating HMAC for viewCopy")
926     }
927     viewCopyEnc = append(viewCopyEnc, viewCopyHMAC...)
928     viewCopyAddr := uuid.New()
929     userlib.DatastoreSet(viewCopyAddr, viewCopyEnc)
930
931     // Step 5: Construct and sign invitation with hybrid encryption
932     viewBytes, err := json.Marshal(fvcopy)
933     if err != nil {
934         return uuid.Nil, errors.New("CreateInvitation: Error marshaling FileView")
935     }
936
937     encryptedKey, encryptedView, err := HybridEncrypt(recipientPK, viewBytes)
938     if err != nil {
939         return uuid.Nil, errors.New("CreateInvitation: Error encrypting FileView: " + err.Error())
940     }
941
942     sig, err := userlib.DSSign(userdata.SignKey, encryptedView)
943     if err != nil {
944         return uuid.Nil, errors.New("CreateInvitation: Error signing invitation")
945     }
946
947     inv := Invitation{
948         EncView:     encryptedView,
949         EncryptedKey: encryptedKey,
950         SenderSig:    sig,
951     }
952
953     // 6. Store invitation and return UUID
954
955     invID := uuid.New()
956     // 加上记录
957     if curFv.PendingInv == nil {
958         curFv.PendingInv = make(map[string]uuid.UUID)
959     }
960     curFv.PendingInv[recipientUsername] = invID
961     curFileList[filename] = curFv
962
963     // 更新用户 fileList
964     err = SaveUserFileList(userFileListID, fileListEncKey, fileListHMACKey, curFileList)
965     if err != nil {
966         return uuid.Nil, errors.New("CreateInvitation: failed to save updated file list")
967     }
```

```
968
969         invBytes, err := json.Marshal(inv)
970         if err != nil {
971             return uuid.Nil, errors.New("CreateInvitation: Error marshaling invitation")
972         }
973         userlib.DatastoreSet(invID, invBytes)
974
975         defer ZeroBytes(fileListEncKey)
976         defer ZeroBytes(fileListHMACKey)
977         defer ZeroBytes(ShareSymKey)
978         defer ZeroBytes(ShareHMACKey)
979         defer ZeroBytes(curFmEncKey)
980         defer ZeroBytes(curFmHMAC)
981         return invID, nil
982 }
983
984 // AcceptInvitation allows a recipient to accept a shared file securely
985 func (userdata *User) AcceptInvitation(senderUsername string, invitationPtr uuid.UUID, filename
    string) (err error) {
986         // Step 1: Verify sender's signature key
987         senderVerifyKey, ok := userlib.KeystoreGet(senderUsername + "_DS")
988         if !ok {
989             return errors.New("AcceptInvitation: sender's signature key not found")
990         }
991
992         // Step 2: Get and parse invitation
993         invByte, exist := userlib.DatastoreGet(invitationPtr)
994
995         if !exist || len(invByte) == 0 {
996             err := errors.New("AcceptInvitation: invitation missing or revoked")
997             fmt.Println("Created error:", err)
998             return err
999         }
1000
1001         var curInv Invitation
1002         err = json.Unmarshal(invByte, &curInv)
1003         if err != nil {
1004             return errors.New("AcceptInvitation: Error unmarshaling invitation")
1005         }
1006         // fmt.Println("AcceptInvitation get curInv", curInv)
1007
1008         if len(curInv.EncView) == 0 || len(curInv.EncryptedKey) == 0 || len(curInv.SenderSig) == 0 {
1009             return errors.New("AcceptInvitation: invitation revoked or malformed")
1010         }
1011
1012         // Step 3: Verify EncView signature
1013         if err := userlib.DSVerify(senderVerifyKey, curInv.EncView, curInv.SenderSig); err != nil {
1014             return errors.New("AcceptInvitation: invalid invitation signature")
1015         }
1016
1017         if curInv.EncryptedKey == nil || curInv.EncView == nil || curInv.SenderSig == nil {
1018             return errors.New("AcceptInvitation: invalid or revoked invitation data")
```

```
1019        }
1020
1021        // Step 4: Decrypt EncView → FileView
1022        _, viewBytes, err := HybridDecrypt(userdata.PrivateKey, curInv.EncryptedKey, curInv.EncView)
1023        if err != nil {
1024            return errors.New("AcceptInvitation: cannot decrypt FileView: " + err.Error())
1025        }
1026
1027        var view FileView
1028        err = json.Unmarshal(viewBytes, &view)
1029        if err != nil {
1030            return errors.New("AcceptInvitation: cannot unmarshal FileView")
1031        }
1032
1033        // Step 5: Verify FileView status
1034        if view.Status != "Share" {
1035            return errors.New("AcceptInvitation: invalid FileView status")
1036        }
1037
1038        //5. Get file list
1039
1040        // 1. Derive keys using helper
1041        fileListEncKey, fileListHMACKey, err := DeriveKeys(userdata.FileKey, []byte("fileListEncKey"),
    []byte("fileListHMACKey"))
1042        if err != nil {
1043            return errors.New("AcceptInvitation: failed to derive file list keys")
1044        }
1045
1046        // 2. Compute UUID of user's file list
1047        UserFileListAddr, err := uuid.FromBytes(userlib.Hash([]byte(userdata.Username + "fileList"))[:16])
1048        if err != nil {
1049            return errors.New("AcceptInvitation: failed to derive UUID")
1050        }
1051
1052        // 3. Fetch from datastore
1053        curFileList, err := LoadUserFileList(UserFileListAddr, fileListEncKey, fileListHMACKey, true)
1054        if err != nil {
1055            return errors.New("AcceptInvitation: failed to load file list")
1056        }
1057
1058        //6. Add FileView to FileList, mark recerived
1059        _, exist = curFileList[filename]
1060        if exist {
1061            return errors.New("AcceptInvitation: File already exist")
1062        } else {
1063            view.Status = "Received"
1064            curFileList[filename] = view
1065        }
1066
1067        //7. Delete invitation info
1068        userlib.DatastoreDelete(invitationPtr)
1069
```

```
1070        //8. Get Filemetadata and Verify FileMetadata integrity
1071        var metadata *FileMetadata
1072        metadata, err = LoadFileMetadata(view.MetadataUUID, view.EncKey, view.HMACKey)
1073        if err != nil {
1074            return errors.New("AcceptInvitation: Failed to load file metadata: " + err.Error())
1075        }
1076
1077        // 9. Verify and update SignedShareList
1078        signedList, err := LoadSignedShareList(metadata.ShareListAddr)
1079        if err != nil {
1080            return errors.New("AcceptInvitation: Failed to load share list")
1081        }
1082        if signedList.List == nil {
1083            signedList.List = make(map[string][]ShareEntry)
1084        }
1085
1086        // Check for duplicate
1087        for _, entry := range signedList.List[senderUsername] {
1088            if entry.Recipient == userdata.Username {
1089                return errors.New("AcceptInvitation: already shared with this user")
1090            }
1091        }
1092
1093        // 10. Add ShareEntry and store share list
1094        newEntry := ShareEntry{
1095            Sender:      senderUsername,
1096            Recipient:   userdata.Username,
1097            FileKey:    userdata.FileKey,
1098            MetadataUUID: view.MetadataUUID,
1099            Filename:    filename,
1100        }
1101        signedList.List[senderUsername] = append(signedList.List[senderUsername], newEntry)
1102
1103        //store the updated share list
1104        err = SaveSignedShareList(metadata.ShareListAddr, signedList)
1105        if err != nil {
1106            return err
1107        }
1108
1109        //11. Store FileMetadata
1110        err = SaveFileMetadata(view.MetadataUUID, metadata, view.EncKey, view.HMACKey)
1111        if err != nil {
1112            return errors.New("AcceptInvitation: Error store UserFileMetadata")
1113        }
1114        //12. Store UserFileList
1115        err = SaveUserFileList(UserFileListAddr, fileListEncKey, fileListHMACKey, curFileList)
1116        if err != nil {
1117            return errors.New("AcceptInvitation: Error store UserFileList")
1118        }
1119        // fmt.Println("FINISH accept: curFileList:", curFileList)
1120        return nil
1121    }
```

```
1122
1123 // RevokeAccess removes the recipient and all of their downstream shared users from access to the file
1124 func (userdata *User) RevokeAccess(filename string, recipientUsername string) error {
1125       //1. Get UserFileList
1126       userFileListID, err := uuid.FromBytes(userlib.Hash([]byte(userdata.Username + "fileList"))[:16])
1127       if err != nil {
1128             return errors.New("RevokeAccess: Failed to derive UUID")
1129       }
1130       //Derive keys for user's file list
1131       fileListEncKey, fileListHMACKey, err := DeriveKeys(userdata.FileKey, []byte("fileListEncKey"),
    []byte("fileListHMACKey"))
1132       if err != nil {
1133             return errors.New("RevokeAccess: Failed to derive file list keys")
1134       }
1135
1136       curFileList, err := LoadUserFileList(userFileListID, fileListEncKey, fileListHMACKey, true)
1137       if err != nil {
1138             return errors.New("RevokeAccess: Failed to load user file list")
1139       }
1140
1141       //2. Check if file exist
1142       fileView, exist := curFileList[filename]
1143       if !exist {
1144             return errors.New("RevokeAccess: File not exist")
1145       }
1146
1147       //3. Load and verify FileMetadata
1148       metadata, err := LoadFileMetadata(fileView.MetadataUUID, fileView.EncKey, fileView.HMACKey)
1149       if err != nil {
1150             return errors.New("RevokeAccess: Failed to load file metadata")
1151       }
1152
1153       //4. Check if the user is the owner of the file
1154       if metadata.Owner != userdata.Username {
1155             return errors.New("RevokeAccess: user is not the owner")
1156       }
1157
1158       // Load SignedShareList
1159       signedList, err := LoadSignedShareList(metadata.ShareListAddr)
1160       if err != nil {
1161             return err
1162       }
1163
1164       // 确保 List 不为 nil
1165       if signedList.List == nil {
1166             signedList.List = make(map[string][]ShareEntry) // 初始化空 map
1167       }
1168
1169       // BFS to find users to revoke
1170       senderEntries, ok := signedList.List[userdata.Username]
1171       if !ok {
1172             // 撤回未被接受的邀请
```

```
1173            fileList, err := LoadUserFileList(userFileListID, fileListEncKey, fileListHMACKey, true)
1174            if err != nil {
1175                return errors.New("RevokeAccess: Failed to load user file list")
1176            }
1177
1178            fileView, ok := fileList[filename]
1179            if !ok {
1180                return errors.New("RevokeAccess: File not found in file list")
1181            }
1182
1183            invMap := fileView.PendingInv
1184            if invMap == nil {
1185                return errors.New("RevokeAccess: No pending invitation map found")
1186            }
1187
1188            invID, exists := invMap[recipientUsername]
1189            if exists {
1190                userlib.DatastoreDelete(invID)
1191
1192                delete(invMap, recipientUsername)
1193                fileView.PendingInv = invMap
1194                fileView.Status = "Own"
1195                fileList[filename] = fileView
1196
1197                err := SaveUserFileList(userFileListID, fileListEncKey, fileListHMACKey, fileList)
1198                if err != nil {
1199                    return errors.New("RevokeAccess: Failed to save file list after deleting pending invite")
1200                }
1201
1202                _, ok := userlib.DatastoreGet(invID)
1203                if ok {
1204                    userlib.DebugMsg("RevokeAccess Deletion failed: invitation still exists in Datastore!")
1205                } else {
1206                    userlib.DebugMsg("RevokeAccess Deletion successful: invitation no longer exists.")
1207                }
1208                return nil
1209            }
1210
1211            return errors.New("RevokeAccess: No share or pending invitation found for this user")
1212        }
1213
1214        var originalShare ShareEntry
1215        found := false
1216        for _, entry := range senderEntries {
1217            if entry.Recipient == recipientUsername {
1218                originalShare = entry
1219                found = true
1220                break
1221            }
1222        }
1223
1224        if !found {
```

```
1225            return errors.New("RevokeAccess: Recipient not found in share entries")
1226        }
1227
1228        // Step 7: BFS 遍历 ShareList，构造 revokeUsers 和 validUsers 列表
1229        revokeUsers := make(map[string][]ShareEntry) // 要撤销的用户及其 shareEntry
1230        remainUsers := make(map[string][]ShareEntry) // 仍然有效的用户及其 shareEntry
1231
1232        // Step 7.1: 将 originalShare 添加到撤销列表（由 owner → recipient 的那一条）
1233        revokeUsers[userdata.Username] = []ShareEntry{originalShare}
1234
1235        // Step 7.2: BFS 队列初始化
1236        queue := []string{recipientUsername}
1237        for len(queue) > 0 {
1238            current := queue[0]
1239            queue = queue[1:]
1240
1241            // 遍历 current 用户分享出去的所有记录（即其下游）
1242            for _, entry := range signedList.List[current] {
1243                recipient := entry.Recipient
1244
1245                // 如果 recipient 还未被撤销
1246                if _, alreadyRevoked := revokeUsers[recipient]; !alreadyRevoked {
1247                    // 标记 sender（current）撤销的分享记录
1248                    revokeUsers[entry.Sender] = append(revokeUsers[entry.Sender], entry)
1249
1250                    // 将 recipient 加入队列，继续扩展
1251                    queue = append(queue, recipient)
1252                }
1253            }
1254        }
1255
1256        for sender, entries := range signedList.List {
1257            // Case 1: sender 是当前用户，不能发给 recipientUsername
1258            if sender == userdata.Username {
1259                var validEntries []ShareEntry
1260                for _, entry := range entries {
1261                    if entry.Recipient != recipientUsername {
1262                        validEntries = append(validEntries, entry)
1263                    }
1264                }
1265                if len(validEntries) > 0 {
1266                    remainUsers[sender] = validEntries
1267                }
1268                continue
1269            }
1270
1271            // Case 2: sender 没有被撤销，直接保留
1272            if _, revoked := revokeUsers[sender]; !revoked {
1273                remainUsers[sender] = entries
1274            }
1275        }
1276
```

```
1277        // 2. 删除所有被撤销用户的 FileView 条目
1278        for user := range revokeUsers {
1279            userFileListUUID, _ := uuid.FromBytes(userlib.Hash([]byte(user + "fileList"))[:16])
1280            userFileListBytes, ok := userlib.DatastoreGet(userFileListUUID)
1281            if !ok {
1282                return errors.New("RevokeAccess: UserFileList not exist")
1283            }
1284            if len(userFileListBytes) < 64 {
1285                return errors.New("RevokeAccess: UserFileList length < 64")
1286            }
1287
1288            var recipientFileKey []byte
1289            var recipientFileName string
1290            for _, entries := range signedList.List {
1291                for _, e := range entries {
1292                    if e.Recipient == user {
1293                        recipientFileKey = e.FileKey
1294                        recipientFileName = e.Filename
1295                    }
1296                }
1297            }
1298            if recipientFileKey == nil {
1299                continue
1300            }
1301            if recipientFileName == "" {
1302                continue
1303            }
1304
1305            fileListEncKey, fileListHMACKey, err := DeriveKeys(recipientFileKey, []byte("fileListEncKey"),
    []byte("fileListHMACKey"))
1306            if err != nil {
1307                return errors.New("RevokeAccess: Error generate fileListEncKey")
1308            }
1309
1310            var fileList map[string]FileView
1311            fileList, err = LoadUserFileList(userFileListUUID, fileListEncKey, fileListHMACKey, true)
1312            if err != nil {
1313                return errors.New("RevokeAccess: Error load UserFileList")
1314            }
1315
1316            _, exist = fileList[recipientFileName]
1317
1318            if !exist {
1319                return errors.New("RevokeAccess: Revoking File entry not exist")
1320            }
1321            delete(fileList, filename)
1322
1323            // 更新 UserFileList
1324            err = SaveUserFileList(userFileListUUID, fileListEncKey, fileListHMACKey, fileList)
1325            if err != nil {
1326                return errors.New("RevokeAccess: Error save UserFileList")
1327            }
```

```
1328        }
1329        // 3. 重新生成文件内容 Chunk
1330        content, err := userdata.LoadFile(filename)
1331        if err != nil {
1332            return errors.New("RevokeAccess: Error loading file content")
1333        }
1334
1335        newEncKey, newHMACKey, _ := DeriveKeys(userlib.RandomBytes(16), []byte("fileEncKey"),
    []byte("fileHMACKey"))
1336
1337        newHead := uuid.New()
1338        newTail := uuid.New()
1339        newChunk := FileChunk{
1340            Data: content,
1341            Next: newTail,
1342        }
1343
1344        err = SaveFileChunk(newEncKey, newHMACKey, newHead, &newChunk)
1345        if err != nil {
1346            return err
1347        }
1348
1349        var newSignedList SignedShareList
1350        // 4. 生成新 ShareList
1351        if len(remainUsers) == 0 {
1352            newSignedList = SignedShareList{
1353                List: make(map[string][]ShareEntry), // 显式初始化
1354            }
1355        } else {
1356            newSignedList = SignedShareList{
1357                List: remainUsers,
1358            }
1359        }
1360
1361        newShareListAddr := uuid.New()
1362        err = SaveSignedShareList(newShareListAddr, &newSignedList)
1363        if err != nil {
1364            return err
1365        }
1366
1367        // 5. 创建新 Metadata
1368        newMetadata := FileMetadata{
1369            Owner:        userdata.Username,
1370            FileName:     filename,
1371            HeadPtr:      newHead,
1372            TailPtr:      newTail,
1373            NumberChunk:   1,
1374            FileEncKey:    newEncKey,
1375            HMACKey:      newHMACKey,
1376            ShareListAddr: newShareListAddr,
1377            Version:      metadata.Version + 1,
1378        }
```

```
1379        metadataEncKey, metadataHMACKey, err := DeriveKeys(userlib.RandomBytes(16),
    []byte("filemetadataEncKey"), []byte("filemetadataHMACKey"))
1380        if err != nil {
1381            return err
1382        }
1383        newMetadataUUID := uuid.New()
1384
1385        err = SaveFileMetadata(newMetadataUUID, &newMetadata, metadataEncKey, metadataHMACKey)
1386        if err != nil {
1387            return err
1388        }
1389
1390        // Update valid users' file list
1391        for _, entries := range remainUsers {
1392            for _, e := range entries {
1393                recipient := e.Recipient
1394                recipientMasterKey := e.FileKey
1395                recipientFileName := e.Filename
1396
1397                fileListEncKey, fileListHMACKey, err := DeriveKeys(recipientMasterKey,
    []byte("fileListEncKey"), []byte("fileListHMACKey"))
1398                userListUUID, _ := uuid.FromBytes(userlib.Hash([]byte(recipient + "fileList"))[:16])
1399                if err != nil {
1400                    return errors.New("RevokeAccess: Error generate UUID")
1401                }
1402
1403                var fileList map[string]FileView
1404                fileList, err = LoadUserFileList(userListUUID, fileListEncKey, fileListHMACKey, true)
1405                if err != nil {
1406                    return errors.New("RevokeAccess: Error load UserFileList")
1407                }
1408
1409                view, exist := fileList[recipientFileName]
1410                if !exist {
1411                    return errors.New("RevokeAccess: Valid File entry not exist")
1412                }
1413
1414                view.MetadataUUID = newMetadataUUID
1415                view.EncKey = metadataEncKey
1416                view.HMACKey = metadataHMACKey
1417                view.Status = "Received"
1418                fileList[recipientFileName] = view
1419                fileList[filename] = view
1420
1421                err = SaveUserFileList(userListUUID, fileListEncKey, fileListHMACKey, fileList)
1422                if err != nil {
1423                    return errors.New("RevokeAccess: Error save valid user UserFileList")
1424                }
1425            }
1426        }
1427
1428        // 6. 更新原始拥有者的 FileView
```

```
1429        ownerListUUID, _ := uuid.FromBytes(userlib.Hash([]byte(userdata.Username + "fileList"))[:16])
1430        ownerFileListEncKey, ownerFileListHMACKey, err := DeriveKeys(userdata.FileKey,
    []byte("fileListEncKey"), []byte("fileListHMACKey"))
1431        if err != nil {
1432            return errors.New("RevokeAccess: Failed to derive file list keys")
1433        }
1434        var ownerfileList map[string]FileView
1435        ownerfileList, err = LoadUserFileList(ownerListUUID, ownerFileListEncKey, ownerFileListHMACKey,
    true)
1436        if err != nil {
1437            return errors.New("RevokeAccess: Failed to load owner file list")
1438        }
1439
1440        ownerfileList[filename] = FileView{
1441            MetadataUUID: newMetadataUUID,
1442            EncKey:      metadataEncKey,
1443            HMACKey:     metadataHMACKey,
1444            Status:      "Own",
1445        }
1446
1447        // Store updated FileView
1448        err = SaveUserFileList(ownerListUUID, ownerFileListEncKey, ownerFileListHMACKey, ownerfileList)
1449        if err != nil {
1450            return err
1451        }
1452
1453        // 7. 删除旧 ShareList、Chunk 链、旧 Metadata
1454        ptr := metadata.HeadPtr
1455        for ptr != metadata.TailPtr {
1456            chunkBytes, ok := userlib.DatastoreGet(ptr)
1457            if !ok || len(chunkBytes) < 64 {
1458                break
1459            }
1460            var chunk FileChunk
1461            plain := userlib.SymDec(metadata.FileEncKey, chunkBytes[:len(chunkBytes)-64])
1462            err := json.Unmarshal(plain, &chunk)
1463            if err != nil {
1464                break
1465            }
1466            next := chunk.Next
1467            userlib.DatastoreDelete(ptr)
1468            ptr = next
1469        }
1470        userlib.DatastoreDelete(metadata.TailPtr)
1471        userlib.DatastoreDelete(metadata.ShareListAddr)
1472        userlib.DatastoreDelete(fileView.MetadataUUID)
1473
1474        defer ZeroBytes(newEncKey)
1475        defer ZeroBytes(newHMACKey)
1476        defer ZeroBytes(metadataEncKey)
1477        defer ZeroBytes(metadataHMACKey)
1478        defer ZeroBytes(fileListEncKey)
```

```
1479        defer ZeroBytes(fileListHMACKey)
1480
1481        return nil
1482    }
1483
```

client/client_unittest.go Download

```go
package client

////////////////////////////////////////////////
//                                            //
// Everything in this file will NOT be graded!!! //
//                                            //
////////////////////////////////////////////////

// In this unit tests file, you can write white-box unit tests on your implementation.
// These are different from the black-box integration tests in client_test.go,
// because in this unit tests file, you can use details specific to your implementation.

// For example, in this unit tests file, you can access struct fields and helper methods
// that you defined, but in the integration tests (client_test.go), you can only access
// the 8 functions (StoreFile, LoadFile, etc.) that are common to all implementations.

// In this unit tests file, you can write InitUser where you would write client.InitUser in the
// integration tests (client_test.go). In other words, the "client." in front is no longer needed.

import (
	userlib "github.com/cs161-staff/project2-userlib"
	"testing"
)

import (
	_ "encoding/hex"
	_ "errors"
	. "github.com/onsi/ginkgo/v2"
	. "github.com/onsi/gomega"
	_ "strconv"
	_ "strings"
)

func TestSetupAndExecution(t *testing.T) {
	RegisterFailHandler(Fail)
	RunSpecs(t, "Client Unit Tests")
}

var _ = Describe("Client Unit Tests", func() {

	BeforeEach(func() {
		userlib.DatastoreClear()
		userlib.KeystoreClear()
	})

	Describe("Unit Tests", func() {
		Specify("Basic Test: Check that the Username field is set for a new user", func() {
			userlib.DebugMsg("Initializing user Alice.")
			// Note: In the integration tests (client_test.go) this would need to
```

```
50                  // be client.InitUser, but here (client_unittests.go) you can write InitUser.
51                  alice, err := InitUser("alice", "password")
52                  Expect(err).To(BeNil())
53
54                  // Note: You can access the Username field of the User struct here.
55                  // But in the integration tests (client_test.go), you cannot access
56                  // struct fields because not all implementations will have a username field.
57                  Expect(alice.Username).To(Equal("alice"))
58              })
59      })
60  })
61
```

client_test/attacker_helpers.go

Download

```go
package client_test

// import (
//     // Some imports use an underscore to prevent the compiler from complaining
//     // about unused imports.

//     _ "encoding/hex"
//     "encoding/json"
//     _ "encoding/json"
//     _ "errors"
//     _ "strconv"
//     _ "strings"

//     "github.com/google/uuid"
//     _ "github.com/google/uuid"

//     userlib "github.com/cs161-staff/project2-userlib"

//     "github.com/cs161-staff/project2-starter-code/client"
// )

// ////////////////////////////
// // 🧪 模拟结构体（映射你的 UserMetadata）
// ////////////////////////////

// type SimulatedUserMetadata struct {
//     EncryptedPrivateKey       []byte
//     PublicKey                 []byte
//     SignatureKey              []byte
//     RootFilePointer           []byte
//     FileMappings              map[string]uuid.UUID
//     FileMappingsEncryptedKeys map[string][]byte // 仅用于测试攻击者记忆 EncryptedKey
// }

// ////////////////////////////
// // 🧠 从 Datastore 提取用户 Metadata（模拟攻击者记忆的内容）
// ////////////////////////////

// func extractUserMetadata(user *client.User) SimulatedUserMetadata {
//     metaUUIDBytes := userlib.Hash([]byte(user.Username + "metadata"))[:16]
//     metaUUID, _ := uuid.FromBytes(metaUUIDBytes)

//     raw, ok := userlib.DatastoreGet(metaUUID)
//     if !ok {
//         panic("UserMetadata not found for user: " + user.Username)
//     }

//     var meta SimulatedUserMetadata
//     err := json.Unmarshal(raw, &meta)
```

```
50  //     if err != nil {
51  //         panic("Failed to parse UserMetadata for user: " + err.Error())
52  //     }
53  //
54  //     return meta
55  // }
56  //
57  // ////////////////////////////
58  // // 🔓 攻击者可读取的模拟接口
59  // ////////////////////////////
60  //
61  // // 获取指定文件的 FileUUID
62  // func attackerGetFileUUID(user *client.User, filename string) uuid.UUID {
63  //     meta := extractUserMetadata(user)
64  //     return meta.FileMappings[filename]
65  // }
66  //
67  // // 获取某文件在接收 invitation 时存储的 EncryptedFileKey（需你在 AcceptInvitation 中记录）
68  // func attackerRememberEncryptedKey(user *client.User, filename string) []byte {
69  //     meta := extractUserMetadata(user)
70  //     return meta.FileMappingsEncryptedKeys[filename]
71  // }
72  //
73  // // 尝试用旧 EncryptedFileKey（模拟 Replay）伪造访问（实际上无法操作底层 key 解密，只能模拟）
74  // func attackerForgeLoadWithOldEncryptedKey(user *client.User, fileUUID uuid.UUID, encryptedKey
    []byte) bool {
75  //     // 模拟攻击者试图访问旧文件路径
76  //     // 实际只能通过 LoadFile 尝试加载旧映射（应该失败）
77  //     _, err := user.LoadFile("fromAlice") // 假设 fromAlice 是 revoked 共享路径
78  //     return err == nil
79  // }
80  //
81  // ////////////////////////////
82  // // 🎯 模拟暴力攻击（猜测 UUID、读取块）
83  // ////////////////////////////
84  //
85  // // 猜测若干 UUID（例如：用 hash-based UUID 结构）
86  // func attackerGuessChunkUUIDs() []uuid.UUID {
87  //     var guesses []uuid.UUID
88  //     base := userlib.Hash([]byte("knownPattern"))
89  //
90  //     for i := 0; i < 5; i++ {
91  //         uuidGuess, _ := uuid.FromBytes(base[i : i+16])
92  //         guesses = append(guesses, uuidGuess)
93  //     }
94  //
95  //     return guesses
96  // }
97  //
98  // // 尝试读取某个 UUID 是否存在（模拟暴力探测 datastore）
99  // func attackerTryReadChunk(chunkUUID uuid.UUID) []byte {
100 //     data, ok := userlib.DatastoreGet(chunkUUID)
```

```
//     if ok {
//         return data
//     }
//     return nil
// }

// // 修改攻击者已知的 UUID 上的数据（模拟篡改 datastore 内容）
// func attackerDirectlyModifyChunk(user *client.User) bool {
//     meta := extractUserMetadata(user)

//     for _, fileUUID := range meta.FileMappings {
//         data, ok := userlib.DatastoreGet(fileUUID)
//         if ok && len(data) > 0 {
//             data[0] ^= 0xFF // Bit flip 攻击
//             userlib.DatastoreSet(fileUUID, data)
//             return true
//         }
//     }

//     return false
// }

```

client_test/client_test.go

Download

```go
package client_test

// You MUST NOT change these default imports.  ANY additional imports may
// break the autograder and everyone will be sad.

import (
	// Some imports use an underscore to prevent the compiler from complaining
	// about unused imports.

	_ "encoding/hex"
	_ "encoding/json"
	_ "errors"
	"math/rand"
	_ "strconv"
	_ "strings"
	"testing"
	"time"

	"github.com/google/uuid"
	_ "github.com/google/uuid"

	// A "dot" import is used here so that the functions in the ginko and gomega
	// modules can be used without an identifier. For example, Describe() and
	// Expect() instead of ginko.Describe() and gomega.Expect().
	. "github.com/onsi/ginkgo/v2"
	. "github.com/onsi/gomega"

	userlib "github.com/cs161-staff/project2-userlib"

	"github.com/cs161-staff/project2-starter-code/client"
)

func TestSetupAndExecution(t *testing.T) {
	RegisterFailHandler(Fail)
	RunSpecs(t, "Client Tests")
}

// ================================================
// Global Variables (feel free to add more!)
// ================================================
const defaultPassword = "password"
const emptyString = ""
const contentOne = "Bitcoin is Nick's favorite "
const contentTwo = "digital "
const contentThree = "cryptocurrency!"
const AESKeySizeBytes = 16
const AESBlockSizeBytes = 16

// ================================================
```

```
50  // Describe(...) blocks help you organize your tests
51  // into functional categories. They can be nested into
52  // a tree-like structure.
53  // ================================================
54
55  var _ = Describe("Client Tests", func() {
56
57      // A few user declarations that may be used for testing. Remember to initialize these before you
58      // attempt to use them!
59
60      var alice *client.User
61      var bob *client.User
62      var charles *client.User
63      // var doris *client.User
64      // var eve *client.User
65      // var frank *client.User
66      // var grace *client.User
67      // var horace *client.User
68      // var ira *client.User
69
70      // These declarations may be useful for multi-session testing.
71      var alicePhone *client.User
72      var aliceLaptop *client.User
73      var aliceDesktop *client.User
74
75      var err error
76
77      // A bunch of filenames that may be useful.
78      aliceFile := "aliceFile.txt"
79      bobFile := "bobFile.txt"
80      charlesFile := "charlesFile.txt"
81      // dorisFile := "dorisFile.txt"
82      // eveFile := "eveFile.txt"
83      // frankFile := "frankFile.txt"
84      // graceFile := "graceFile.txt"
85      // horaceFile := "horaceFile.txt"
86      // iraFile := "iraFile.txt"
87
88      BeforeEach(func() {
89          // This runs before each test within this Describe block (including nested tests).
90          // Here, we reset the state of Datastore and Keystore so that tests do not interfere with each
    other.
91          // We also initialize
92          userlib.DatastoreClear()
93          userlib.KeystoreClear()
94      })
95
96      Describe("Basic Tests", func() {
97
98          Specify("Basic Test: Testing InitUser/GetUser on a single user.", func() {
99              userlib.DebugMsg("Initializing user Alice.")
100             alice, err = client.InitUser("alice", defaultPassword)
```

```
101                 Expect(err).To(BeNil())
102
103                 userlib.DebugMsg("Getting user Alice.")
104                 aliceLaptop, err = client.GetUser("alice", defaultPassword)
105                 Expect(err).To(BeNil())
106                 Expect(aliceLaptop.Username).To(Equal("alice")) // or any other property/method
107             })
108
109             Specify("Basic Test: Testing Single User Store/Load/Append.", func() {
110                 userlib.DebugMsg("Initializing user Alice.")
111                 alice, err = client.InitUser("alice", defaultPassword)
112                 Expect(err).To(BeNil())
113
114                 userlib.DebugMsg("Storing file data: %s", contentOne)
115                 err = alice.StoreFile(aliceFile, []byte(contentOne))
116                 Expect(err).To(BeNil())
117
118                 userlib.DebugMsg("Appending file data: %s", contentTwo)
119                 err = alice.AppendToFile(aliceFile, []byte(contentTwo))
120                 Expect(err).To(BeNil())
121
122                 userlib.DebugMsg("Appending file data: %s", contentThree)
123                 err = alice.AppendToFile(aliceFile, []byte(contentThree))
124                 Expect(err).To(BeNil())
125
126                 userlib.DebugMsg("Loading file...")
127                 data, err := alice.LoadFile(aliceFile)
128                 Expect(err).To(BeNil())
129                 Expect(data).To(Equal([]byte(contentOne + contentTwo + contentThree)))
130             })
131
132             Specify("Basic Test: Testing Create/Accept Invite Functionality with multiple users and multiple
    instances.", func() {
133                 userlib.DebugMsg("Initializing users Alice (aliceDesktop) and Bob.")
134                 aliceDesktop, err = client.InitUser("alice", defaultPassword)
135                 Expect(err).To(BeNil())
136
137                 bob, err = client.InitUser("bob", defaultPassword)
138                 Expect(err).To(BeNil())
139
140                 userlib.DebugMsg("Getting second instance of Alice - aliceLaptop")
141                 aliceLaptop, err = client.GetUser("alice", defaultPassword)
142                 Expect(err).To(BeNil())
143
144                 userlib.DebugMsg("aliceDesktop storing file %s with content: %s", aliceFile, contentOne)
145                 err = aliceDesktop.StoreFile(aliceFile, []byte(contentOne))
146                 Expect(err).To(BeNil())
147
148                 userlib.DebugMsg("aliceLaptop creating invite for Bob.")
149                 invite, err := aliceLaptop.CreateInvitation(aliceFile, "bob")
150                 // _, err := aliceLaptop.CreateInvitation(aliceFile, "bob")
151                 Expect(err).To(BeNil())
```

```
152
153                 userlib.DebugMsg("Bob accepting invite from Alice under filename %s.", bobFile)
154                 err = bob.AcceptInvitation("alice", invite, bobFile)
155                 Expect(err).To(BeNil())
156
157                 userlib.DebugMsg("Bob appending to file %s, content: %s", bobFile, contentTwo)
158                 err = bob.AppendToFile(bobFile, []byte(contentTwo))
159                 Expect(err).To(BeNil())
160
161                 userlib.DebugMsg("aliceDesktop appending to file %s, content: %s", aliceFile,
    contentThree)
162                 err = aliceDesktop.AppendToFile(aliceFile, []byte(contentThree))
163                 Expect(err).To(BeNil())
164
165                 userlib.DebugMsg("Checking that aliceDesktop sees expected file data.")
166                 data, err := aliceDesktop.LoadFile(aliceFile)
167                 Expect(err).To(BeNil())
168                 Expect(data).To(Equal([]byte(contentOne + contentTwo + contentThree)))
169
170                 userlib.DebugMsg("Checking that aliceLaptop sees expected file data.")
171                 data, err = aliceLaptop.LoadFile(aliceFile)
172                 Expect(err).To(BeNil())
173                 Expect(data).To(Equal([]byte(contentOne + contentTwo + contentThree)))
174
175                 userlib.DebugMsg("Checking that Bob sees expected file data.")
176                 data, err = bob.LoadFile(bobFile)
177                 Expect(err).To(BeNil())
178                 Expect(data).To(Equal([]byte(contentOne + contentTwo + contentThree)))
179
180                 userlib.DebugMsg("Getting third instance of Alice - alicePhone.")
181                 alicePhone, err = client.GetUser("alice", defaultPassword)
182                 Expect(err).To(BeNil())
183
184                 userlib.DebugMsg("Checking that alicePhone sees Alice's changes.")
185                 data, err = alicePhone.LoadFile(aliceFile)
186                 Expect(err).To(BeNil())
187                 Expect(data).To(Equal([]byte(contentOne + contentTwo + contentThree)))
188             })
189
190             Specify("Basic Test: Testing Revoke Functionality", func() {
191                 userlib.DebugMsg("Initializing users Alice, Bob, and Charlie.")
192                 alice, err = client.InitUser("alice", defaultPassword)
193                 Expect(err).To(BeNil())
194
195                 bob, err = client.InitUser("bob", defaultPassword)
196                 Expect(err).To(BeNil())
197
198                 charles, err = client.InitUser("charles", defaultPassword)
199                 Expect(err).To(BeNil())
200
201                 userlib.DebugMsg("Alice storing file %s with content: %s", aliceFile, contentOne)
202                 alice.StoreFile(aliceFile, []byte(contentOne))
```

```
203
204                 userlib.DebugMsg("Alice creating invite for Bob for file %s, and Bob accepting invite under
    name %s.", aliceFile, bobFile)
205                 invite, err := alice.CreateInvitation(aliceFile, "bob")
206                 Expect(err).To(BeNil())
207
208                 err = bob.AcceptInvitation("alice", invite, bobFile)
209                 Expect(err).To(BeNil())
210
211                 userlib.DebugMsg("Checking that Alice can still load the file.")
212                 data, err := alice.LoadFile(aliceFile)
213                 Expect(err).To(BeNil())
214                 Expect(data).To(Equal([]byte(contentOne)))
215
216                 userlib.DebugMsg("Checking that Bob can load the file.")
217                 data, err = bob.LoadFile(bobFile)
218                 Expect(err).To(BeNil())
219                 Expect(data).To(Equal([]byte(contentOne)))
220
221                 userlib.DebugMsg("Bob creating invite for Charles for file %s, and Charlie accepting invite
    under name %s.", bobFile, charlesFile)
222                 invite, err = bob.CreateInvitation(bobFile, "charles")
223                 Expect(err).To(BeNil())
224
225                 err = charles.AcceptInvitation("bob", invite, charlesFile)
226                 Expect(err).To(BeNil())
227
228                 userlib.DebugMsg("Checking that Bob can load the file.")
229                 data, err = bob.LoadFile(bobFile)
230                 Expect(err).To(BeNil())
231                 Expect(data).To(Equal([]byte(contentOne)))
232
233                 userlib.DebugMsg("Checking that Charles can load the file.")
234                 data, err = charles.LoadFile(charlesFile)
235                 Expect(err).To(BeNil())
236                 Expect(data).To(Equal([]byte(contentOne)))
237
238                 userlib.DebugMsg("Alice revoking Bob's access from %s.", aliceFile)
239                 err = alice.RevokeAccess(aliceFile, "bob")
240                 Expect(err).To(BeNil())
241
242                 userlib.DebugMsg("Checking that Alice can still load the file.")
243                 data, err = alice.LoadFile(aliceFile)
244                 Expect(err).To(BeNil())
245                 Expect(data).To(Equal([]byte(contentOne)))
246
247                 userlib.DebugMsg("Checking that Bob/Charles lost access to the file.")
248                 _, err = bob.LoadFile(bobFile)
249                 Expect(err).ToNot(BeNil())
250
251                 _, err = charles.LoadFile(charlesFile)
252                 Expect(err).ToNot(BeNil())
```

```
253
254                 userlib.DebugMsg("Checking that the revoked users cannot append to the file.")
255                 err = bob.AppendToFile(bobFile, []byte(contentTwo))
256                 Expect(err).ToNot(BeNil())
257
258                 err = charles.AppendToFile(charlesFile, []byte(contentTwo))
259                 Expect(err).ToNot(BeNil())
260             })
261
262     })
263
264     Describe("InitUser Tests", func() {
265
266         Specify("InitUser Tests: initialize a new user", func() {
267             userlib.DebugMsg("Initializing user Alice.")
268             alice, err = client.InitUser("alice", defaultPassword)
269             Expect(err).To(BeNil(), "Failed to init user Alice")
270             Expect(alice).NotTo(BeNil(), "User Alice should not be nil")
271         })
272
273         Specify("InitUser Tests: return an error if the username is empty", func() {
274             // 输入的用户名为空
275             userlib.DebugMsg("Initializing with empty username.")
276             _, err = client.InitUser("", defaultPassword)
277             Expect(err).NotTo(BeNil(), "Expected an error for empty username")
278             // Expect(err.Error()).To(ContainSubstring("username cannot be empty"), "Error message
    should indicate empty username")
279         })
280
281         Specify("InitUser Tests: return an error if the user already exists", func() {
282             userlib.DebugMsg("Initializing user Alice.")
283             alice, err = client.InitUser("alice", defaultPassword)
284             Expect(err).To(BeNil(), "Failed to init user Alice")
285
286             // 用户名已存在
287             userlib.DebugMsg("Initializing again with username Alice.")
288             _, err = client.InitUser("alice", defaultPassword)
289             Expect(err).NotTo(BeNil(), "Expected an error for duplicate user")
290             // Expect(err.Error()).To(ContainSubstring("user already exists"), "Error message should
    indicate duplicate user")
291         })
292
293     })
294
295     Describe("GetUser Tests", func() {
296
297         // 输出的用户数据是否正确
298         Specify("GetUser Tests: retrieve an existing user", func() {
299             userlib.DebugMsg("Initializing user Alice.")
300             alice, err = client.InitUser("alice", defaultPassword)
301             Expect(err).To(BeNil(), "Failed to init user Alice")
302
```

```
303                 userlib.DebugMsg("Getting user Alice.")
304                 aliceLaptop, err := client.GetUser("alice", defaultPassword)
305                 Expect(err).To(BeNil(), "Failed to retrieve user Alice")
306                 Expect(aliceLaptop).NotTo(BeNil(), "Retrieved user should not be nil")
307                 Expect(aliceLaptop.Username).To(Equal("alice"), "Retrieved user should have the correct
    username")
308             })
309
310             // 用户不存在
311             Specify("GetUser Tests: return an error if the user does not exist", func() {
312                 userlib.DebugMsg("Getting user Bob.")
313                 _, err = client.GetUser("bob", defaultPassword)
314                 Expect(err).NotTo(BeNil(), "Expected an error for non-existent user")
315                 // Expect(err.Error()).To(ContainSubstring("user not found"), "Error message should
    indicate user not found")
316             })
317
318             // 密码错误
319             Specify("GetUser Tests: return an error if the password is incorrect", func() {
320                 userlib.DebugMsg("Initializing user Alice.")
321                 alice, err = client.InitUser("alice", defaultPassword)
322                 Expect(err).To(BeNil(), "Failed to init user Alice")
323
324                 _, err = client.GetUser("alice", "wrongpassword")
325                 Expect(err).NotTo(BeNil(), "Expected an error for incorrect password")
326                 // Expect(err.Error()).To(ContainSubstring("incorrect password"), "Error message should
    indicate incorrect password")
327             })
328
329             // 输入的用户名为空
330             Specify("GetUser Tests: return an error if the username is empty", func() {
331                 userlib.DebugMsg("Input empty username.")
332                 _, err = client.GetUser("", defaultPassword)
333                 Expect(err).NotTo(BeNil(), "Expected an error for empty username")
334                 // Expect(err.Error()).To(ContainSubstring("username cannot be empty"), "Error message
    should indicate empty username")
335             })
336
337         })
338
339         Describe("MultiDevice Tests(6/6)", func() {
340             Specify("Basic Test: Testing Single User Store/Load/Append.", func() {
341                 userlib.DebugMsg("Initializing user Alice.")
342                 alice, err = client.InitUser("alice", defaultPassword)
343                 Expect(err).To(BeNil())
344
345                 userlib.DebugMsg("Storing file data: %s", contentOne)
346                 err = alice.StoreFile(aliceFile, []byte(contentOne))
347                 Expect(err).To(BeNil())
348
349                 userlib.DebugMsg("Appending file data: %s", contentTwo)
350                 err = alice.AppendToFile(aliceFile, []byte(contentTwo))
```

```
351                 Expect(err).To(BeNil())
352
353                 userlib.DebugMsg("Appending file data: %s", contentThree)
354                 err = alice.AppendToFile(aliceFile, []byte(contentThree))
355                 Expect(err).To(BeNil())
356
357                 userlib.DebugMsg("Loading file...")
358                 data, err := alice.LoadFile(aliceFile)
359                 Expect(err).To(BeNil())
360                 Expect(data).To(Equal([]byte(contentOne + contentTwo + contentThree)))
361             })
362
363             // 1. 数据不同步(当用户在一台设备上更新文件后，另一台设备无法立即看到最新数据)
364             Specify("Multiple devices should see latest file updates", func() {
365                 userlib.DebugMsg("Initializing user Alice.")
366                 _, err := client.InitUser("alice", defaultPassword)
367                 Expect(err).To(BeNil(), "Failed to init user Alice")
368
369                 userlib.DebugMsg("Retrieving user Alice on another device.")
370                 aliceLaptop, err := client.GetUser("alice", defaultPassword)
371                 Expect(err).To(BeNil(), "Failed to retrieve user Alice on laptop")
372
373                 alicePhone, err := client.GetUser("alice", defaultPassword)
374                 Expect(err).To(BeNil(), "Failed to retrieve user Alice on phone")
375
376                 // 电脑上存储文件
377                 userlib.DebugMsg("Alice stores a file on laptop.")
378                 err = aliceLaptop.StoreFile(aliceFile, []byte(contentOne))
379                 Expect(err).To(BeNil(), "Failed to store file on laptop")
380
381                 // 手机上加载文件
382                 userlib.DebugMsg("Alice loads the file on phone.")
383                 data, err := alicePhone.LoadFile(aliceFile)
384                 Expect(err).To(BeNil(), "Failed to load file on phone")
385                 Expect(data).To(Equal([]byte(contentOne)), "Phone should see latest file update")
386
387                 // 手机上追加内容
388                 userlib.DebugMsg("Alice appends content to file on phone.")
389                 err = alicePhone.AppendToFile(aliceFile, []byte(contentTwo))
390                 Expect(err).To(BeNil(), "Failed to append file on phone")
391
392                 // 电脑上加载更新后的文件
393                 userlib.DebugMsg("Alice loads the file on laptop.")
394                 data, err = aliceLaptop.LoadFile(aliceFile)
395                 Expect(err).To(BeNil(), "Failed to load file on laptop")
396                 Expect(data).To(Equal([]byte(contentOne+contentTwo)), "Laptop should see appended
    content from phone")
397             })
398
399             // 2. 并发冲突(两个设备同时修改文件)
400             Specify("Concurrent modifications should not cause data loss", func() {
401                 userlib.DebugMsg("Initializing user Alice.")
```

```
402             _, err := client.InitUser("alice", defaultPassword)
403             Expect(err).To(BeNil(), "Failed to init user Alice")
404
405             userlib.DebugMsg("Retrieving user Alice on another device.")
406             aliceLaptop, err := client.GetUser("alice", defaultPassword)
407             Expect(err).To(BeNil(), "Failed to retrieve user Alice on laptop")
408
409             alicePhone, err := client.GetUser("alice", defaultPassword)
410             Expect(err).To(BeNil(), "Failed to retrieve user Alice on phone")
411
412             // 电脑上存储文件
413             userlib.DebugMsg("Alice stores a file on laptop.")
414             err = aliceLaptop.StoreFile(aliceFile, []byte(contentOne))
415             Expect(err).To(BeNil(), "Failed to store file on laptop")
416
417             //两个设备同时追加同一文件
418             userlib.DebugMsg("Both devices append to the same file concurrently.")
419             err = aliceLaptop.AppendToFile(aliceFile, []byte(contentOne))
420             err = alicePhone.AppendToFile(aliceFile, []byte(contentTwo))
421
422             // 等待 100ms 确保并发执行
423             time.Sleep(100 * time.Millisecond)
424
425             userlib.DebugMsg("Alice loads the file on laptop.")
426             data, err := aliceLaptop.LoadFile(aliceFile)
427             Expect(err).To(BeNil(), "Failed to load file after concurrent updates")
428             Expect(data).To(ContainSubstring(contentOne), "Laptop's update is missing")
429             Expect(data).To(ContainSubstring(contentTwo), "Phone's update is missing")
430
431             userlib.DebugMsg("Alice loads the file on phone.")
432             data, err = alicePhone.LoadFile(aliceFile)
433             Expect(err).To(BeNil(), "Failed to load file after concurrent updates")
434             Expect(data).To(ContainSubstring(contentOne), "Laptop's update is missing")
435             Expect(data).To(ContainSubstring(contentTwo), "Phone's update is missing")
436         })
437
438         // 3. 设备缓存问题
439         Specify("Devices should not use outdated cached data", func() {
440             userlib.DebugMsg("Initializing user Alice.")
441             _, err := client.InitUser("alice", defaultPassword)
442             Expect(err).To(BeNil(), "Failed to init user Alice")
443
444             userlib.DebugMsg("Retrieving user Alice on another device.")
445             aliceLaptop, err := client.GetUser("alice", defaultPassword)
446             Expect(err).To(BeNil(), "Failed to retrieve user Alice on laptop")
447
448             alicePhone, err := client.GetUser("alice", defaultPassword)
449             Expect(err).To(BeNil(), "Failed to retrieve user Alice on phone")
450
451             // 电脑上存储文件
452             userlib.DebugMsg("Alice stores a file on laptop.")
453             err = aliceLaptop.StoreFile(aliceFile, []byte(contentOne))
```

```
454             Expect(err).To(BeNil(), "Failed to store file on laptop")
455
456             // 手机上加载文件
457             userlib.DebugMsg("Alice loads the file on phone.")
458             data, err := alicePhone.LoadFile(aliceFile)
459             Expect(err).To(BeNil(), "Failed to load file on phone")
460             Expect(data).To(Equal([]byte(contentOne)), "Phone should see latest file update")
461
462             // 电脑上存储文件
463             userlib.DebugMsg("Alice change the file on laptop.")
464             err = aliceLaptop.StoreFile(aliceFile, []byte(contentTwo))
465             Expect(err).To(BeNil(), "Failed to stores file on laptop")
466
467             // 确保手机读取到的是最新值
468             userlib.DebugMsg("Alice loads the file on laptop.")
469             data, err = alicePhone.LoadFile(aliceFile)
470             Expect(err).To(BeNil(), "Failed to load updated file")
471             Expect(data).To(Equal([]byte(contentTwo)), "Phone loaded outdated cached data")
472         })
473
474         // 4. 断网后的数据一致性
475         Specify("Offline edits should merge correctly after reconnecting", func() {
476             userlib.DebugMsg("Initializing user Alice.")
477             _, err := client.InitUser("alice", defaultPassword)
478             Expect(err).To(BeNil(), "Failed to init user Alice")
479
480             userlib.DebugMsg("Retrieving user Alice on another device.")
481             aliceLaptop, err := client.GetUser("alice", defaultPassword)
482             Expect(err).To(BeNil(), "Failed to retrieve user Alice on laptop")
483
484             alicePhone, err := client.GetUser("alice", defaultPassword)
485             Expect(err).To(BeNil(), "Failed to retrieve user Alice on phone")
486
487             // 电脑上存储文件
488             userlib.DebugMsg("Alice stores a file on laptop.")
489             err = aliceLaptop.StoreFile(aliceFile, []byte(contentOne))
490             Expect(err).To(BeNil(), "Failed to store file on laptop")
491
492             // 手机上加载文件
493             userlib.DebugMsg("Alice loads the file on phone.")
494             data, err := alicePhone.LoadFile(aliceFile)
495             Expect(err).To(BeNil(), "Failed to load file on phone")
496             Expect(data).To(Equal([]byte(contentOne)), "Phone should see latest file update")
497
498             // 模拟断网并修改
499             userlib.DebugMsg("Alice change the file on laptop.")
500             err = aliceLaptop.StoreFile(aliceFile, []byte(contentTwo))
501             Expect(err).To(BeNil(), "Failed to store offline edit on laptop")
502
503             userlib.DebugMsg("Alice append the file on phone.")
504             alicePhone.AppendToFile(aliceFile, []byte(contentThree))
505             Expect(err).To(BeNil(), "Failed to append offline edit on phone")
```

```
506
507                 // 重新同步
508                 content, err := aliceLaptop.LoadFile(aliceFile)
509                 Expect(err).To(BeNil(), "Failed to load after reconnection")
510                 userlib.DebugMsg("data: %s", content)
511                 Expect(content).To(SatisfyAny(
512                     Equal([]byte(contentTwo+contentThree)),
513                     Equal([]byte(contentThree+contentTwo)),
514                 ), "Offline edits not merged correctly")
515             })
516
517             // 5. 多次登录后的数据完整性
518             Specify("Logging in from multiple devices should retain data", func() {
519                 userlib.DebugMsg("Initializing user Alice.")
520                 _, err := client.InitUser("alice", defaultPassword)
521                 Expect(err).To(BeNil(), "Failed to init user Alice")
522
523                 userlib.DebugMsg("Retrieving user Alice.")
524                 aliceLaptop, err := client.GetUser("alice", defaultPassword)
525                 Expect(err).To(BeNil(), "Failed to retrieve user Alice on laptop")
526
527                 userlib.DebugMsg("Alice stores a file on laptop.")
528                 err = aliceLaptop.StoreFile(aliceFile, []byte(contentOne))
529                 Expect(err).To(BeNil(), "Failed to store file on laptop")
530
531                 // 重新登录
532                 aliceLaptop, _ = client.GetUser("alice", defaultPassword)
533                 err = aliceLaptop.AppendToFile(aliceFile, []byte(contentTwo))
534                 Expect(err).To(BeNil(), "Failed to append file after relogin")
535
536                 data, err := aliceLaptop.LoadFile(aliceFile)
537                 Expect(err).To(BeNil(), "Failed to load session file")
538                 Expect(data).To(Equal([]byte(contentOne+contentTwo)), "Session data lost after relogin")
539             })
540
541     })
542
543     Describe("AppendToFile Efficiency Tests(0/3)", func() {
544         BeforeEach(func() {
545             userlib.DebugMsg("Running BeforeEach...")
546             userlib.DatastoreClear()
547             userlib.DatastoreResetBandwidth()
548             // 添加延迟确保清除完成
549             time.Sleep(100 * time.Millisecond)
550
551             alice, err = client.InitUser("alice", defaultPassword)
552             // userlib.DebugMsg("InitUser result: %v, error: %v\n", alice, err)
553             Expect(err).To(BeNil())
554         })
555
556         Specify("success - Appending small data should use minimal bandwidth", func() {
557             // 存储初始文件内容
```

```
558                 userlib.DebugMsg("Storing initial file 'testfile' with content 'Hello'.")
559                 err = alice.StoreFile("testfile", []byte("Hello"))
560                 Expect(err).To(BeNil())
561
562                 appendContent1 := []byte(" World")
563
564                 // 第一次追加小数据
565                 before1 := userlib.DatastoreGetBandwidth()
566                 userlib.DebugMsg("Appending ' World' to 'testfile'.")
567                 err = alice.AppendToFile("testfile", appendContent1)
568                 Expect(err).To(BeNil())
569
570                 finalBandwidth1 := userlib.DatastoreGetBandwidth()
571                 bandwidthUsed1 := finalBandwidth1 - before1
572
573                 // 验证带宽使用仅与追加数据大小成比例
574                 userlib.DebugMsg("Bandwidth used for append: %d bytes.", bandwidthUsed1)
575                 Expect(bandwidthUsed1).To(BeNumerically("<=", len(appendContent1)+3000)) // 允许一个
    小的常数开销
576             })
577
578             Specify("success - Appending large data should use bandwidth proportional to append size",
    func() {
579                 // 存储初始文件内容
580                 userlib.DebugMsg("Storing initial file 'largefile' with content 'Start'.")
581                 err = alice.StoreFile("largefile", []byte("Start"))
582                 Expect(err).To(BeNil())
583
584                 // 记录追加操作前的带宽使用情况
585                 initialBandwidth := userlib.DatastoreGetBandwidth()
586
587                 // 定义两个不同大小的追加数据
588                 smallAppendContent := make([]byte, 1*1024*1024)  // 1MB
589                 largeAppendContent := make([]byte, 10*1024*1024) // 10MB
590
591                 // 记录第一次追加（小文件）的带宽使用
592                 userlib.DebugMsg("Appending 1MB of data to 'largefile'.")
593                 err = alice.AppendToFile("largefile", smallAppendContent)
594                 Expect(err).To(BeNil())
595                 bandwidthSmall := userlib.DatastoreGetBandwidth() - initialBandwidth
596
597                 // 追加大数据
598                 beforeLarge := userlib.DatastoreGetBandwidth() // 重新获取追加前的带宽
599                 userlib.DebugMsg("Appending 10MB of data to 'largefile'.")
600                 err = alice.AppendToFile("largefile", largeAppendContent)
601                 Expect(err).To(BeNil())
602                 bandwidthLarge := userlib.DatastoreGetBandwidth() - beforeLarge
603
604                 // // 验证带宽使用仅与追加数据大小成比例
605                 // userlib.DebugMsg("Bandwidth used for large append: %d bytes.", bandwidthSmall)
606                 // Expect(bandwidthUsed).To(BeNumerically("<=", len(largeAppendContent)+3000)) // 允许
    一个小的常数开销
```

```
607
608                 // 验证比例关系：bandwidthLarge / bandwidthSmall ≈ 10MB / 1MB = 10
609                 ratio := float64(bandwidthLarge) / float64(bandwidthSmall)
610                 expectedRatio := float64(len(largeAppendContent)) / float64(len(smallAppendContent))
611
612                 userlib.DebugMsg("Bandwidth ratio (large/small): %.2f (expected ≈ %.2f)", ratio,
    expectedRatio)
613                 Expect(ratio).To(BeNumerically("~", expectedRatio, 0.1)) // 允许10%误差
614             })
615
616             Specify("success - Multiple small appends should not cause increasing bandwidth usage",
    func() {
617                 // 存储初始文件内容
618                 userlib.DebugMsg("Storing initial file 'multitest' with content 'Init'.")
619                 err = alice.StoreFile("multitest", []byte("Init"))
620                 Expect(err).To(BeNil())
621
622                 // 定义要追加的小数据块
623                 appendContent := []byte("A")
624                 numAppends := 100
625                 lastbandwidth := 0
626                 bandwidthUsed := 0
627
628                 for i := 0; i < numAppends; i++ {
629                     // 记录每次追加操作前的带宽使用情况
630                     initialBandwidth := userlib.DatastoreGetBandwidth()
631
632                     // 追加小数据
633                     userlib.DebugMsg("Appending 'A' to 'multitest', iteration %d.", i+1)
634                     err = alice.AppendToFile("multitest", appendContent)
635                     Expect(err).To(BeNil())
636
637                     // 计算追加操作后的带宽使用情况
638                     finalBandwidth := userlib.DatastoreGetBandwidth()
639                     bandwidthUsed = finalBandwidth - initialBandwidth
640
641                     if i != 0 {
642                         // 验证每次追加的带宽使用保持一致
643                         userlib.DebugMsg("Bandwidth used for append %d: %d bytes.", i+1,
    bandwidthUsed)
644                         Expect(lastbandwidth).To(BeNumerically("~", bandwidthUsed, 100))
645                     }
646
647                     lastbandwidth = bandwidthUsed
648                 }
649             })
650     })
651     Describe("Namespacing Tests(1/3)", func() {
652
653             BeforeEach(func() {
654                 // 初始化用户Alice
655                 userlib.DebugMsg("Initializing user Alice.")
```

```
656             alice, err = client.InitUser("alice", defaultPassword)
657             Expect(err).To(BeNil())
658 
659             // 初始化用户Bob
660             userlib.DebugMsg("Initializing user Bob.")
661             bob, err = client.InitUser("bob", defaultPassword)
662             Expect(err).To(BeNil())
663 
664             // 确保数据存储在每个测试前被清空
665             userlib.DatastoreClear()
666         })
667 
668         Specify("1- Different users can have files with the same name without conflict", func() {
669             // Alice存储名为"shared.txt"的文件
670             userlib.DebugMsg("Alice stores a file named 'shared.txt' with content 'Alice's content'.")
671             err = alice.StoreFile("shared.txt", []byte("Alice's content"))
672             Expect(err).To(BeNil())
673 
674             // Bob存储同名文件"shared.txt"
675             userlib.DebugMsg("Bob stores a file named 'shared.txt' with content 'Bob's content'.")
676             err = bob.StoreFile("shared.txt", []byte("Bob's content"))
677             Expect(err).To(BeNil())
678 
679             // 验证Alice的文件内容
680             userlib.DebugMsg("Alice loads 'shared.txt' and expects to see her own content.")
681             content, err := alice.LoadFile("shared.txt")
682             Expect(err).To(BeNil())
683             Expect(content).To(Equal([]byte("Alice's content")))
684 
685             // 验证Bob的文件内容
686             userlib.DebugMsg("Bob loads 'shared.txt' and expects to see his own content.")
687             content, err = bob.LoadFile("shared.txt")
688             Expect(err).To(BeNil())
689             Expect(content).To(Equal([]byte("Bob's content")))
690         })
691 
692         Specify("2- Overwriting a file does not affect other users' files with the same name", func() {
693             // Alice存储名为"notes.txt"的文件
694             userlib.DebugMsg("Alice stores a file named 'notes.txt' with content 'Initial notes'.")
695             err = alice.StoreFile("notes.txt", []byte("Initial notes"))
696             Expect(err).To(BeNil())
697 
698             // Bob存储同名文件"notes.txt"
699             userlib.DebugMsg("Bob stores a file named 'notes.txt' with content 'Bob's notes'.")
700             err = bob.StoreFile("notes.txt", []byte("Bob's notes"))
701             Expect(err).To(BeNil())
702 
703             // Alice覆盖她的"notes.txt"文件
704             userlib.DebugMsg("Alice overwrites 'notes.txt' with new content 'Updated notes'.")
705             err = alice.StoreFile("notes.txt", []byte("Updated notes"))
706             Expect(err).To(BeNil())
707 
```

```
708                 // 验证Alice的文件内容
709                 userlib.DebugMsg("Alice loads 'notes.txt' and expects to see 'Updated notes'.")
710                 content, err := alice.LoadFile("notes.txt")
711                 Expect(err).To(BeNil())
712                 Expect(content).To(Equal([]byte("Updated notes")))
713
714                 // 验证Bob的文件内容未受影响
715                 userlib.DebugMsg("Bob loads 'notes.txt' and expects to see his original content 'Bob's
    notes'.")
716                 content, err = bob.LoadFile("notes.txt")
717                 Expect(err).To(BeNil())
718                 Expect(content).To(Equal([]byte("Bob's notes")))
719             })
720
721             Specify("3- failed -Appending to a file does not affect other users' files with the same name",
    func() {
722                 // Alice存储名为"diary.txt"的文件
723                 userlib.DebugMsg("Alice stores a file named 'diary.txt' with content 'Day 1: Sunny'.")
724                 err = alice.StoreFile("diary.txt", []byte("Day 1: Sunny"))
725                 Expect(err).To(BeNil())
726
727                 // Bob存储同名文件"diary.txt"
728                 userlib.DebugMsg("Bob stores a file named 'diary.txt' with content 'Entry 1: Work'.")
729                 err = bob.StoreFile("diary.txt", []byte("Entry 1: Work"))
730                 Expect(err).To(BeNil())
731
732                 // Alice追加内容到她的"diary.txt"文件
733                 userlib.DebugMsg("Alice appends ' Day 2: Rainy' to her 'diary.txt'.")
734                 err = alice.AppendToFile("diary.txt", []byte(" Day 2: Rainy"))
735                 Expect(err).To(BeNil())
736
737                 // 验证Alice的文件内容
738                 userlib.DebugMsg("Alice loads 'diary.txt' and expects to see 'Day 1: Sunny Day 2: Rainy'.")
739                 content, err := alice.LoadFile("diary.txt")
740                 Expect(err).To(BeNil())
741                 Expect(content).To(Equal([]byte("Day 1: Sunny Day 2: Rainy")))
742
743                 // 验证Bob的文件内容未受影响
744                 userlib.DebugMsg("Bob loads 'diary.txt' and expects to see his original content 'Entry 1:
    Work'.")
745                 content, err = bob.LoadFile("diary.txt")
746                 Expect(err).To(BeNil())
747                 Expect(content).To(Equal([]byte("Entry 1: Work")))
748             })
749         })
750
751         Describe("File Operations Tests(2/5)", func() {
752             BeforeEach(func() {
753                 userlib.DebugMsg("Running BeforeEach...")
754                 userlib.DatastoreClear()
755                 // 添加延迟确保清除完成
756                 time.Sleep(100 * time.Millisecond)
```

```
757         })
758
759         Specify("1- StoreFile creates a new file and LoadFile retrieves its content", func() {
760             // 初始化用户Alice
761             userlib.DebugMsg("Initializing user Alice.")
762             alice, err = client.InitUser("alice", defaultPassword)
763             Expect(err).To(BeNil())
764             // Alice存储名为"document.txt"的文件
765             userlib.DebugMsg("Alice stores a file named 'document.txt' with content 'Hello, World!'.")
766             err = alice.StoreFile("document.txt", []byte("Hello, World!"))
767             Expect(err).To(BeNil())
768
769             // 验证Alice的文件内容
770             userlib.DebugMsg("Alice loads 'document.txt' and expects to see 'Hello, World!'.")
771             content, err := alice.LoadFile("document.txt")
772             Expect(err).To(BeNil())
773             Expect(content).To(Equal([]byte("Hello, World!")))
774         })
775
776         Specify("2- StoreFile overwrites existing file content", func() {
777             // Alice存储名为"notes.txt"的文件
778             // 初始化用户Alice
779             userlib.DebugMsg("Initializing user Alice.")
780             alice, err = client.InitUser("alice", defaultPassword)
781             Expect(err).To(BeNil())
782             userlib.DebugMsg("Alice stores a file named 'notes.txt' with content 'Initial content'.")
783             err = alice.StoreFile("notes.txt", []byte("Initial content"))
784             Expect(err).To(BeNil())
785
786             // Alice覆盖"notes.txt"的内容
787             userlib.DebugMsg("Alice overwrites 'notes.txt' with new content 'Updated content'.")
788             err = alice.StoreFile("notes.txt", []byte("Updated content"))
789             Expect(err).To(BeNil())
790
791             // 验证Alice的文件内容
792             userlib.DebugMsg("Alice loads 'notes.txt' and expects to see 'Updated content'.")
793             content, err := alice.LoadFile("notes.txt")
794             Expect(err).To(BeNil())
795             Expect(content).To(Equal([]byte("Updated content")))
796         })
797
798         Specify("3- LoadFile returns an error for non-existent files", func() {
799             // 尝试加载不存在的文件
800             // 初始化用户Alice
801             userlib.DebugMsg("Initializing user Alice.")
802             alice, err = client.InitUser("alice", defaultPassword)
803             Expect(err).To(BeNil())
804             userlib.DebugMsg("Alice attempts to load 'missing.txt' and expects an error.")
805             content, err := alice.LoadFile("missing.txt")
806             Expect(err).ToNot(BeNil())
807             Expect(content).To(BeNil())
808         })
```

```
809
810         Specify("4-failed AppendToFile adds content to the end of an existing file", func() {
811             // 初始化用户Alice
812             userlib.DebugMsg("Initializing user Alice.")
813             alice, err = client.InitUser("alice", defaultPassword)
814             Expect(err).To(BeNil())
815             // Alice存储名为"journal.txt"的文件
816             userlib.DebugMsg("Alice stores a file named 'journal.txt' with content 'Day 1: Sunny'.")
817             err = alice.StoreFile("journal.txt", []byte("Day 1: Sunny"))
818             Expect(err).To(BeNil())
819
820             // Alice追加内容到"journal.txt"
821             userlib.DebugMsg("Alice appends ' Day 2: Rainy' to 'journal.txt'.")
822             err = alice.AppendToFile("journal.txt", []byte(" Day 2: Rainy"))
823             Expect(err).To(BeNil())
824
825             // 验证Alice的文件内容
826             userlib.DebugMsg("Alice loads 'journal.txt' and expects to see 'Day 1: Sunny Day 2:
    Rainy'.")
827             content, err := alice.LoadFile("journal.txt")
828             Expect(err).To(BeNil())
829             Expect(content).To(Equal([]byte("Day 1: Sunny Day 2: Rainy")))
830         })
831
832         Specify("5-failed AppendToFile returns an error when the file does not exist", func() {
833             // 初始化用户Alice
834             userlib.DebugMsg("Initializing user Alice.")
835             alice, err = client.InitUser("alice", defaultPassword)
836             Expect(err).To(BeNil())
837             // 尝试向不存在的文件追加内容
838             userlib.DebugMsg("Alice attempts to append to 'nonexistent.txt' and expects an error.")
839             err = alice.AppendToFile("nonexistent.txt", []byte("Some content"))
840             Expect(err).ToNot(BeNil())
841         })
842         Specify("6-bigFile", func() {
843             userlib.DebugMsg("Initializing user Alice.")
844             alice, err = client.InitUser("alice2", defaultPassword)
845             Expect(err).To(BeNil())
846             alice.StoreFile("aliceFile", userlib.RandomBytes(10000))
847             alice.AppendToFile("aliceFile", userlib.RandomBytes(10000))
848
849             file, err1 := alice.LoadFile("aliceFile")
850             userlib.DebugMsg("file1:\n", file)
851             Expect(err).To(BeNil())
852             userlib.DebugMsg("err1:\n", err1)
853
854             err2 := alice.AppendToFile("aliceFile", userlib.RandomBytes(10000))
855             userlib.DebugMsg("err2:\n", err2)
856             Expect(err).To(BeNil())
857         })
858     })
859
```

```
860     Describe("File Sharing Tests(5/5)", func() {
861
862         Specify("1-Test: Comprehensive Create/Accept Invitation Test with Integrity Check", func() {
863             alice, err = client.InitUser("alice", defaultPassword)
864             Expect(err).To(BeNil())
865
866             bob, err = client.InitUser("bob", defaultPassword)
867             Expect(err).To(BeNil())
868
869             charles, err = client.InitUser("charles", defaultPassword)
870             Expect(err).To(BeNil())
871
872             // Step 1: Alice 存储数据
873             userlib.DebugMsg("Alice storing file %s with content: %s", aliceFile, contentOne)
874             err = alice.StoreFile(aliceFile, []byte(contentOne))
875             Expect(err).To(BeNil())
876
877             // Step 2: Alice 创建邀请给 Bob
878             invite, err := alice.CreateInvitation(aliceFile, "bob")
879             Expect(err).To(BeNil())
880
881             // Step 3: Bob 接受邀请并给出新文件名
882             userlib.DebugMsg("Bob accepting invitation with filename %s", bobFile)
883             err = bob.AcceptInvitation("alice", invite, bobFile)
884             Expect(err).To(BeNil())
885
886             // Step 4: 验证 Bob 的数据是否与 Alice 一致
887             data, err := bob.LoadFile(bobFile)
888             Expect(err).To(BeNil())
889             Expect(data).To(Equal([]byte(contentOne)))
890
891             // Step 5: Alice 追加新数据
892             userlib.DebugMsg("Alice appending data: %s", contentTwo)
893             err = alice.AppendToFile(aliceFile, []byte(contentTwo))
894             Expect(err).To(BeNil())
895
896             // Step 6: 验证 Bob 的数据是否自动更新
897             data, err = bob.LoadFile(bobFile)
898             Expect(err).To(BeNil())
899             Expect(data).To(Equal([]byte(contentOne + contentTwo)))
900
901             // Step 7: Bob 创建邀请给 charles
902             inviteForCharles, err := bob.CreateInvitation(bobFile, "charles")
903             Expect(err).To(BeNil())
904
905             // Step 8: charles 接受邀请
906             err = charles.AcceptInvitation("bob", inviteForCharles, charlesFile)
907             Expect(err).To(BeNil())
908
909             // Step 9: 验证 charles 也能看到最新数据
910             data, err = charles.LoadFile(charlesFile)
911             Expect(err).To(BeNil())
```

```
912             Expect(data).To(Equal([]byte(contentOne + contentTwo)))
913
914             // Step 10: Alice 再次追加数据，确保 charles /Bob 均可查看
915             userlib.DebugMsg("Alice appending final content: %s", contentThree)
916             err = alice.AppendToFile(aliceFile, []byte(contentThree))
917             Expect(err).To(BeNil())
918
919             // Step 11: 验证 Bob/Charlie 均能查看到最终数据
920             data, err = bob.LoadFile(bobFile)
921             Expect(err).To(BeNil())
922             Expect(data).To(Equal([]byte(contentOne + contentTwo + contentThree)))
923
924             data, err = charles.LoadFile(charlesFile)
925             Expect(err).To(BeNil())
926             Expect(data).To(Equal([]byte(contentOne + contentTwo + contentThree)))
927
928             // Step 12: 检查无效邀请 (恶意伪造的 invitationPtr)
929             fakeInvite := uuid.New()
930             err = charles.AcceptInvitation("alice", fakeInvite, "fakeFile")
931             Expect(err).ToNot(BeNil()) // 错误预期
932
933             // Step 13: 检查已撤销的邀请
934             err = alice.RevokeAccess(aliceFile, "bob")
935             Expect(err).To(BeNil())
936
937             _, err = bob.LoadFile(bobFile)
938             Expect(err).ToNot(BeNil()) // Bob 应该失去访问权限
939
940             _, err = charles.LoadFile(charlesFile)
941             Expect(err).ToNot(BeNil()) // Charlie 也应失去访问权限
942         })
943
944         // Specify("4-Security Test: Prevent Circular Sharing", func() {
945         //     alice, err := client.InitUser("alice", defaultPassword)
946         //     Expect(err).To(BeNil())
947
948         //     bob, err := client.InitUser("bob", defaultPassword)
949         //     Expect(err).To(BeNil())
950
951         //     // Step 1: Alice 创建并存储文件
952         //     err = alice.StoreFile("file.txt", []byte("Circular Sharing Test"))
953         //     Expect(err).To(BeNil())
954
955         //     // Step 2: Alice 创建邀请并分享给 Bob
956         //     inviteBob, err := alice.CreateInvitation("file.txt", "bob")
957         //     Expect(err).To(BeNil())
958         //     err = bob.AcceptInvitation("alice", inviteBob, bobFile)
959         //     Expect(err).To(BeNil())
960
961         //     // Step 3: Bob 尝试将该文件再次分享回 Alice
962         //     _, err = bob.CreateInvitation(bobFile, "alice")
963
```

```
964             //     // Step 4: 验证系统拒绝循环分享
965             //     Expect(err).ToNot(BeNil()) // 应失败
966             // })
967
968             Specify("6-Stress Test: Large File Handling", func() {
969                 alice, err := client.InitUser("alice", defaultPassword)
970                 Expect(err).To(BeNil())
971
972                 bob, err := client.InitUser("bob", defaultPassword)
973                 Expect(err).To(BeNil())
974
975                 // Step 1: 创建一个 5MB 的大文件并随机填充数据
976                 largeData := make([]byte, 5*1024*1024) // 5MB
977                 _, err = rand.Read(largeData)        // 使用随机数据来模拟真实大文件
978                 Expect(err).To(BeNil())
979
980                 // Step 2: Alice 存储超大文件
981                 userlib.DebugMsg("[Step 2] Alice storing large file (5MB)")
982                 err = alice.StoreFile("largeFile.txt", largeData)
983                 Expect(err).To(BeNil())
984
985                 // Step 3: 验证 Alice 是否可以成功加载该超大文件
986                 userlib.DebugMsg("[Step 3] Alice loading large file (5MB)")
987                 loadedData, err := alice.LoadFile("largeFile.txt")
988                 Expect(err).To(BeNil())
989                 Expect(loadedData).To(Equal(largeData)) // 数据完整性检查
990
991                 // Step 4: Alice 创建邀请并分享给 Bob
992                 inviteBob, err := alice.CreateInvitation("largeFile.txt", "bob")
993                 Expect(err).To(BeNil())
994                 err = bob.AcceptInvitation("alice", inviteBob, "bobLargeFile.txt")
995                 Expect(err).To(BeNil())
996
997                 // Step 5: 验证 Bob 能够正确加载共享的超大文件
998                 userlib.DebugMsg("[Step 5] Bob loading shared large file (5MB)")
999                 sharedData, err := bob.LoadFile("bobLargeFile.txt")
1000                Expect(err).To(BeNil())
1001                Expect(sharedData).To(Equal(largeData)) // 数据完整性检查
1002
1003                // Step 6: Bob 向文件追加更多数据
1004                additionalData := []byte(" - Bob's Contribution")
1005                err = bob.AppendToFile("bobLargeFile.txt", additionalData)
1006                Expect(err).To(BeNil())
1007
1008                // Step 7: Alice 验证文件已正确追加
1009                finalData := append(largeData, additionalData...)
1010                loadedData, err = alice.LoadFile("largeFile.txt")
1011                Expect(err).To(BeNil())
1012                Expect(loadedData).To(Equal(finalData)) // 确保完整性
1013
1014                // Step 8: Alice 撤销 Bob 的访问权限
1015                err = alice.RevokeAccess("largeFile.txt", "bob")
```

```
1016                Expect(err).To(BeNil())
1017
1018                // Step 9: Bob 尝试再次访问应失败
1019                _, err = bob.LoadFile("bobLargeFile.txt")
1020                Expect(err).ToNot(BeNil()) // Bob 的访问应被拒绝
1021            })
1022
1023        })
1024
1025        Describe("RevokeAccess Tests(3/4)", func() {
1026
1027            Specify("7.1 - Replay of Old Invitation Should Fail", func() {
1028                alice, err = client.InitUser("alice", defaultPassword)
1029                Expect(err).To(BeNil())
1030
1031                bob, err = client.InitUser("bob", defaultPassword)
1032                Expect(err).To(BeNil())
1033
1034                err = alice.StoreFile(aliceFile, []byte("top secret"))
1035                Expect(err).To(BeNil())
1036
1037                invite, err := alice.CreateInvitation(aliceFile, "bob")
1038                Expect(err).To(BeNil())
1039
1040                err = bob.AcceptInvitation("alice", invite, bobFile)
1041                Expect(err).To(BeNil())
1042
1043                err = alice.RevokeAccess(aliceFile, "bob")
1044                Expect(err).To(BeNil())
1045
1046                err = bob.AcceptInvitation("alice", invite, "replayed")
1047                Expect(err).ToNot(BeNil())
1048            })
1049            Specify("7.5 - Revoke Subtree Recursively", func() {
1050                alice, _ = client.InitUser("alice", defaultPassword)
1051                bob, _ = client.InitUser("bob", defaultPassword)
1052                charles, _ = client.InitUser("charles", defaultPassword)
1053
1054                _ = alice.StoreFile(aliceFile, []byte("secret"))
1055                inviteBob, _ := alice.CreateInvitation(aliceFile, "bob")
1056                _ = bob.AcceptInvitation("alice", inviteBob, bobFile)
1057
1058                inviteDave, _ := bob.CreateInvitation(bobFile, "charles")
1059                _ = charles.AcceptInvitation("bob", inviteDave, charlesFile)
1060
1061                _ = alice.RevokeAccess(aliceFile, "bob")
1062
1063                _, err := charles.LoadFile(charlesFile)
1064                Expect(err).ToNot(BeNil())
1065            })
1066            Specify("7.6 - Non-Revoked User Access Unaffected", func() {
1067                alice, _ = client.InitUser("alice", defaultPassword)
```

```
1068                userlib.DebugMsg("Initializing user Alice.")
1069                bob, _ = client.InitUser("bob", defaultPassword)
1070                charles, _ = client.InitUser("charles", defaultPassword)
1071
1072                _ = alice.StoreFile(aliceFile, []byte("stable"))
1073                inviteBob, _ := alice.CreateInvitation(aliceFile, "bob")
1074                _ = bob.AcceptInvitation("alice", inviteBob, bobFile)
1075
1076                inviteCharlie, _ := alice.CreateInvitation(aliceFile, "charles")
1077                _ = charles.AcceptInvitation("alice", inviteCharlie, charlesFile)
1078
1079                userlib.DebugMsg("charlesFile before revoke:", charlesFile)
1080
1081                _ = alice.RevokeAccess(aliceFile, "bob")
1082                data, err := charles.LoadFile(charlesFile)
1083                Expect(err).To(BeNil())
1084                Expect(data).To(Equal([]byte("stable")))
1085        })
1086
1087        Specify("7.8 - Revoked User Cannot Create New Invitations", func() {
1088                // 初始化用户
1089                alice, err := client.InitUser("alice", defaultPassword)
1090                Expect(err).To(BeNil())
1091
1092                bob, err := client.InitUser("bob", defaultPassword)
1093                Expect(err).To(BeNil())
1094
1095                // Alice 存储文件并分享给 Bob
1096                err = alice.StoreFile("file.txt", []byte("Content"))
1097                Expect(err).To(BeNil())
1098                inviteBob, err := alice.CreateInvitation("file.txt", "bob")
1099                Expect(err).To(BeNil())
1100                err = bob.AcceptInvitation("alice", inviteBob, bobFile)
1101                Expect(err).To(BeNil())
1102
1103                // Alice 撤销 Bob 的访问权限
1104                err = alice.RevokeAccess("file.txt", "bob")
1105                Expect(err).To(BeNil())
1106
1107                // 验证 Bob 无法创建新的邀请
1108                _, err = bob.CreateInvitation(bobFile, "charles")
1109                Expect(err).ToNot(BeNil())
1110        })
1111
1112        Specify("7.9 - Test pendingInv not influence revoke", func() {
1113                alice, err = client.InitUser("alice", defaultPassword)
1114                Expect(err).To(BeNil())
1115
1116                bob, err = client.InitUser("bob", defaultPassword)
1117                Expect(err).To(BeNil())
1118
1119                userlib.DebugMsg("Alice storing file %s with content: %s", aliceFile, contentOne)
```

```
1120                alice.StoreFile(aliceFile, []byte(contentOne))
1121
1122                userlib.DebugMsg("Alice creating invite for Bob for file %s, and Bob accepting invite under
    name %s.", aliceFile, bobFile)
1123                invite, err := alice.CreateInvitation(aliceFile, "bob")
1124                Expect(err).To(BeNil())
1125
1126                err = bob.AcceptInvitation("alice", invite, bobFile)
1127                Expect(err).To(BeNil())
1128
1129                userlib.DebugMsg("Alice revoking Bob's access from %s.", aliceFile)
1130                err = alice.RevokeAccess(aliceFile, "bob")
1131                Expect(err).To(BeNil())
1132
1133            })
1134
1135            Specify("7.10 - Revoke before accept invitation", func() {
1136                alice, err = client.InitUser("alice", defaultPassword)
1137                Expect(err).To(BeNil())
1138
1139                bob, err = client.InitUser("bob", defaultPassword)
1140                Expect(err).To(BeNil())
1141
1142                err = alice.StoreFile(aliceFile, []byte(contentOne))
1143                Expect(err).To(BeNil())
1144
1145                userlib.DebugMsg("Alice creating invite for Bob for file %s.", aliceFile)
1146                invite, err := alice.CreateInvitation(aliceFile, "bob")
1147                Expect(err).To(BeNil())
1148
1149                // alice 在bob接受邀请之前就撤销了bob的访问权限
1150                userlib.DebugMsg("Alice revoking Bob's access from %s.", aliceFile)
1151                err = alice.RevokeAccess(aliceFile, "bob")
1152                Expect(err).To(BeNil())
1153
1154                err = bob.AcceptInvitation("alice", invite, bobFile)
1155                Expect(err).ToNot(BeNil())
1156
1157                _, err = bob.LoadFile(bobFile)
1158                Expect(err).ToNot(BeNil())
1159
1160                err = bob.AppendToFile(bobFile, []byte(contentTwo))
1161                Expect(err).ToNot(BeNil())
1162
1163                //bob自己调用storefile创建的一个在自己namespace的文件 和 alice无关
1164                err = bob.StoreFile(bobFile, []byte(contentTwo))
1165                Expect(err).To(BeNil())
1166                var aliceData, bobData []byte
1167                bobData, err = bob.LoadFile(bobFile)
1168                if err == nil {
1169                    Expect(bobData).To(Equal([]byte(contentTwo)))
1170                    Expect(bobData).ToNot(Equal(aliceData))
```

```
1171                        userlib.DebugMsg("Bob's file is not the same as Alice's file.")
1172
1173                    }
1174                })
1175
1176                // Specify("7.7 - UUID Guessing Should Fail", func() {
1177                //     alice, _ = client.InitUser("alice", defaultPassword)
1178                //     bob, _ = client.InitUser("bob", defaultPassword)
1179
1180                //     _ = alice.StoreFile(aliceFile, []byte("hidden"))
1181                //     invite, _ := alice.CreateInvitation(aliceFile, "bob")
1182                //     _ = bob.AcceptInvitation("alice", invite, bobFile)
1183
1184                //     _ = alice.RevokeAccess(aliceFile, "bob")
1185
1186                //     guesses := attackerGuessChunkUUIDs()
1187                //     for _, guess := range guesses {
1188                //         data := attackerTryReadChunk(guess)
1189                //         Expect(data).To(BeNil())
1190                //     }
1191                // })
1192        })
1193    })
1194
```

fileStore.md

Download

1. chunk层次

1. key生成，用来enc chunk data, 存储在metadata里

```go
fileEncKey, fileHMACKey, err := DeriveKeys(userlib.RandomBytes(16), []byte("fileEncKey"), []byte("fileHMACKey"))
```

2. 加密data（不是加密整个chunk）：

```go
encryptedChunkData, chunkHMAC, err := EasyEncrypt(fileEncKey, fileHMACKey, data)
```

这里的chunkHMAC 存储在chunk struct中

3. 存储：

```go
chunkBytes, err := encodejson(fileChunk)
userlib.DatastoreSet(chunkUUID, chunkBytes)
```

2. metadata层

```go
fileMetadata := FileMetadata{
    FileEncKey:    fileEncKey,
    HMACKey:       fileHMACKey,
}
```

1. key生成, 用于加密metadata，存储在fileview中

```go
metadataEncKey, metadataHMACKey, err := DeriveKeys(userlib.RandomBytes(16), []byte("filemetadataEncKey"), []byte("filemetadataHMACKey"))
```

2. 加密整个metadata：

```go
fileMetadataBytes, err := encodejson(fileMetadata)
fileMetaEnc, fileMetaHmac, err := EasyEncrypt(metadataEncKey, metadataHMACKey, fileMetadataBytes)
```

3. 存储

```go
SaveFileMetadata(fileMetadataUUID, fileMetaEnc, fileMetaHmac, userdata.SignKey):
    sig, err := userlib.DSSign(signingKey, userlib.Hash(fileMetaEnc))
    final := append(fileMetaEnc, fileMetaHmac...)
    final = append(final, sig...)
    userlib.DatastoreSet(id, final)
```

```
```

3. fileview层
```go
fileView.EncKey = metadataEncKey
fileView.HMACKey = metadataHMACKey
fileView.MetadataUUID = fileMetadataUUID

fileList[filename] = fileView
```

4. fileList层
```go
fileListEncKey, fileListHMACKey, err := DeriveKeys(userdata.MasterKey, []byte("fileListEncKey"), []byte("fileListHMACKey"))

fileListBytes, _ := encodejson(fileList)
fileListEnc, fileListHMAC, err := EasyEncrypt(fileListEncKey, fileListHMACKey, fileListBytes)

fileListStore := append(fileListEnc, fileListHMAC...)
userlib.DatastoreSet(userFileListID, fileListStore)
```

fileshare.md Download

```
CreateInvitation
开始
 │
 ├── 加载用户元数据 → 验证HMAC → 解密 → 解析为UserMetadata
 │
 ├── 检查文件是否存在 → 获取文件UUID
 │
 ├── 加载文件元数据 → 验证签名
 │
 ├── 解密PKE私钥 → 解密EncryptedFileKey获得fileKey
 │
 ├── 派生共享密钥（deriveKey(fileKey, token)）
 │
 ├── 用接收方公钥加密共享密钥 → 生成EncryptedKey
 │
 ├── 构造Invitation → 签名 → 存储到Datastore
 │
 ├── 更新文件元数据的SharedWith（添加接收方）
 │
 └── 返回邀请UUID

AcceptInvitation
开始
 │
 ├── 加载邀请 → 验证签名（注释部分需取消）
 │
 ├── 解析文件UUID → 加载文件元数据 → 验证所有者签名
 │
 ├── 解密接收方的PKE私钥 → 解密EncryptedKey获得中间密钥（无需二次派生）
 │
 ├── 更新用户元数据的FileMappings（添加文件）
 │
 ├── 更新文件元数据的SharedWith（添加当前用户）
 │
 └── 保存元数据

RevokeAccess
开始
 │
 ├── 加载用户元数据 → 验证文件存在 → 检查所有者权限
 │
 ├── 解密原fileKey → 生成新fileKey → 加密后更新文件元数据
 │
 ├── 遍历所有共享用户：
 │    ├── 根据路径（Parent链）逐级派生新密钥（基于新fileKey和token）
 │    ├── 用用户公钥加密新派生密钥 → 更新SharedWith
 │
 └── 保存文件元数据

```

数据结构设计

1. 文件副本结构 (File Shard)
```go
type FileShard struct {
    ShardID      string         // 副本唯一标识（如"shard1", "shard2"）
    Owner        string         // 所属直接分享者（如B属于A的shard1，C属于A的shard2）
    rootFile     File           // 指向原文件
    DelegateEncryptedKeys map[string][]byte // 次级用户密钥（用户名 → 用该用户公钥加密的密钥）
    Revoked      bool           // 标记是否被撤销
    KeyVersion   uint64         // 密钥版本号（撤销时递增）
}
```

2. 用户权限映射 (User Access Map)
```go
type UserAccess struct {
    Username    string
    ShardID     string   // 指向的副本ID（如D指向shard1）
    Invitation  []byte   // 加密的邀请信息（防止篡改）
}
```

3. 全局文件记录 (Global File Record)
```go
type FileRecord struct {
    Filename    string
    Owner       string              // A
    Shards      map[string]*FileShard  // 所有副本（key为shardID）
    DirectShares map[string]string     // 直接分享的用户→副本ID（如A→B:shard1, A→C:shard2）
}
```

核心操作流程

1. 初始化文件分享 (A → B/C)

创建邀请：
1. 发送方选择文件
2. 用接收方公钥加密FileKey：EncryptedKeyForUserX = PKE_Encrypt(ReceiverPublicKey, (FileKey || FileMetadata.Version))
3. 创建签名：SenderSig = Sign(senderPriv, FileUUID|EncryptedKeyForUserX)
4. 构建新副本``ShareFile``
5. 存储``Invitation``到Datastore（加密存储）
6. 返回 `InvitationUUID` 给 `Sender`，并传送给 `Receiver`（通过外部信道）

接受邀请：
1. 接收方查询Invitation（需身份认证）
2. 验证签名：Verify(senderPub, FileUUID|EncryptedKeyForUserX, SenderSig)
3. 解密获得FileKey和Version：FileKey|V = RSA_Dec(receiverPriv, EncryptedKeyForUserX)
4. 检查V >= 文件当前版本（防版本回滚）
5. 将FileKey绑定到用户元数据：

```
   UserMetadata.FileMappings[FileUUID] = EncryptedKeyForUserX
6. 更新文件SharedWith列表：
   File.SharedWith[receiver] = struct{
      Key:   EncryptedKey,
      Chain: ShareChain
   }
7. 标记Invitation状态为已接受


```go
func ShareFile(owner, rootfile file , receiver string) {
    // 生成新副本（假设A初次分享给B）
    shardID := GenerateShardID()
    newShard := &FileShard{
        ShardID:     shardID,
        Owner:       owner,
        rootFile：    rootfile
        DelegateEncryptedKeys[receiver] = PKE_Encrypt(ReceiverPublicKey, (FileKey ||
FileMetadata.Version))
        Revoked:    false,
        KeyVersion:  fileRecord.GlobalVersion,
    }

    // 记录直接分享关系
    fileRecord.DirectShares[receiver] = shardID
    fileRecord.Shards[shardID] = newShard

    // 创建用户权限
    userAccessMap[receiver] = &UserAccess{
        ShardID:    shardID,
    }
}
```

2. 次级分享 (B → D/E/F)
```go
func DelegateShare(sender, receiver string) {
    // 获取发送方所属副本
    shardID := userAccessMap[sender].ShardID
    shard := fileRecord.Shards[shardID]
    // 添加给次级用户的加密秘钥
    shard.DelegateEncryptedKeys[receiver] = PKE_Encrypt(ReceiverPublicKey, (FileKey ||
FileMetadata.Version))

    // 添加次级用户
    userAccessMap[receiver] = &UserAccess{
        ShardID:    shardID,
    }
}
```

3. 权限撤销 (RevokeAccess)
```

```go
func RevokeAccess(owner, targetUser string) {
    // 1. 验证调用者为所有者且目标为直接分享用户
    if fileRecord.Owner != owner ||
       fileRecord.DirectShares[targetUser] == "" {
        return error
    }

    // 2. 获取目标用户的副本ID
    shardID := fileRecord.DirectShares[targetUser]
    targetShard := fileRecord.Shards[shardID]

    // 3. 标记副本及所有子用户失效
    targetShard.Revoked = true
    targetShard.rootFile = Nil
    targetShard.KeyVersion++
    targetShard.DelegateEncryptedKeys = Null  //删除所有给子用户（BDEF）的加密秘钥

    // 4. 生成新密钥
    newFileKey = AES-CTR-GenerateKey()
}
```


访问控制验证

文件访问前校验
```go
func ValidateAccess(username string) bool {
    // 1. 获取用户指向的副本
    shardID := userAccessMap[username].ShardID
    targetShard := fileRecord.Shards[shardID]

    // 2. 验证副本是否有效
    for targetShard != nil {
        if targetShard.IsRevoked || targetShard.KeyVersion < GetCurrentVersion() {
            return false
        }
    }
    return true
}
```

```go
type User struct {
	Username   string
	RootKey    []byte
	PublicKey  userlib.PKEEncKey
	PrivateKey userlib.PKEDecKey
	SignKey    userlib.DSSignKey
	VerifyKey  userlib.DSVerifyKey
```

```
	FileList   map[string]FileView
}

type FileView struct {
	MetadataUUID uuid.UUID
	EncKey       []byte
	HMACKey      []byte
	Status       string // Own | Shared | Received
}

type FileMetadata struct {
	Owner        string
	FileName     string
	StartAddr    uuid.UUID
	NextAddr     uuid.UUID
	FileEncKey   []byte
	HMACKey      []byte
	ShareListAddr uuid.UUID
	Version      uint64
}

type SharedEntry struct {
	Sender       string
	Recipient    string
	SourceKey    []byte
	MetadataUUID uuid.UUID
}

type Invitation struct {
	EncView   []byte
	SenderSig []byte
}

type ShareList = map[string][]SharedEntry
type FileChunk struct {
	Data     []byte      // 加密后的文件内容（或明文，然后用 FileEncKey 加密）
	Next     uuid.UUID   // 指向下一个 Chunk（或空 UUID 表示终止）
	HMAC     []byte      // HMAC(Data)，防止篡改
}
```

userlib.go

Download

```go
package userlib

import (
	"errors"
	"fmt"
	"log"
	"strings"
	"sync"
	"time"

	"crypto"
	"crypto/aes"
	"crypto/cipher"
	"crypto/hmac"
	"crypto/rand"
	"crypto/rsa"
	"crypto/sha512"

	. "github.com/onsi/ginkgo/v2"

	"github.com/google/uuid"
	"golang.org/x/crypto/argon2"
)

// More info about the UUID type:
// github.com/google/uuid
type UUID = uuid.UUID

// AES block size (in bytes)
// https://pkg.go.dev/crypto/aes
const AESBlockSizeBytes = aes.BlockSize

// AES key size (in bytes)
const AESKeySizeBytes = 16

// Output size (in bytes) of Hash, HMAC, and HashKDF
const HashSizeBytes = sha512.Size

const rsaKeySizeBits = 2048

// UUID size (in bytes)
const UUIDSizeBytes = 16

/*
*********************************************
**        Global Definitions           ***
*********************************************

Here, we declare a number of global data
```

```
structures and types: Keystore/Datastore,
Public/Private Key structures, etc.
*/

type PublicKeyType struct {
	KeyType string
	PubKey  rsa.PublicKey
}

type PrivateKeyType struct {
	KeyType string
	PrivKey rsa.PrivateKey
}

// Bandwidth tracker (for measuring efficient append)
// var datastoreBandwidth = 0
// map[int]*int
var datastoreBandwidth sync.Map

// Datastore and Keystore variables

type keystoreType map[string]PublicKeyType
type datastoreType map[UUID][]byte

// map[int]keystoreType
var datastore sync.Map

// map[int]datastoreType
var keystore sync.Map

// var datastore map[UUID][]byte = make(map[UUID][]byte)
// var keystore map[string]PublicKeyType = make(map[string]PublicKeyType)

type DatastoreEntry struct {
	UUID  string
	Value string
}

func getKeystoreShard() keystoreType {
	pid := CurrentSpecReport().LineNumber()
	shard, _ := keystore.LoadOrStore(pid, make(keystoreType))
	shardMap := shard.(keystoreType)
	return shardMap
}

func getDatastoreShard() datastoreType {
	pid := CurrentSpecReport().LineNumber()
	shard, _ := datastore.LoadOrStore(pid, make(datastoreType))
	shardMap := shard.(datastoreType)
	return shardMap
}

```

```
102 func getDatastoreBandwidthShard() *int {
103     pid := CurrentSpecReport().LineNumber()
104     newBandwidth := 0
105     bandwidth, _ := datastoreBandwidth.LoadOrStore(pid, &newBandwidth)
106     return bandwidth.(*int)
107 }
108
109 /*
110 ******************************************
111 **         Datastore Functions          **
112 **      DatastoreSet, DatastoreGet,     **
113 **    DatastoreDelete, DatastoreClear   **
114 ******************************************
115  */
116
117 // Sets the value in the datastore
118 func datastoreSet(key UUID, value []byte) {
119     // Update bandwidth tracker
120     bandwidth := getDatastoreBandwidthShard()
121     *bandwidth += len(value)
122
123     foo := make([]byte, len(value))
124     copy(foo, value)
125
126     datastoreShard := getDatastoreShard()
127     datastoreShard[key] = foo
128 }
129
130 var DatastoreSet = datastoreSet
131
132 // Returns the value if it exists
133 func datastoreGet(key UUID) (value []byte, ok bool) {
134     datastoreShard := getDatastoreShard()
135     value, ok = datastoreShard[key]
136     if ok && value != nil {
137         // Update bandwidth tracker
138         bandwidth := getDatastoreBandwidthShard()
139         *bandwidth += len(value)
140
141         foo := make([]byte, len(value))
142         copy(foo, value)
143         return foo, ok
144     }
145     return
146 }
147
148 var DatastoreGet = datastoreGet
149
150 // Deletes a key
151 func datastoreDelete(key UUID) {
152     datastoreShard := getDatastoreShard()
153     delete(datastoreShard, key)
```

```
}

var DatastoreDelete = datastoreDelete

// Use this in testing to reset the datastore to empty
func datastoreClear() {
	datastoreShard := getDatastoreShard()
	for k := range datastoreShard {
		delete(datastoreShard, k)
	}
}

var DatastoreClear = datastoreClear

func DatastoreResetBandwidth() {
	bandwidth := getDatastoreBandwidthShard()
	*bandwidth = 0
}

// Get number of bytes uploaded/downloaded to/from Datastore.
func DatastoreGetBandwidth() int {
	bandwidth := getDatastoreBandwidthShard()
	return *bandwidth
}

// Use this in testing to reset the keystore to empty
func keystoreClear() {
	keystoreShard := getKeystoreShard()
	for k := range keystoreShard {
		delete(keystoreShard, k)
	}
}

var KeystoreClear = keystoreClear

// Sets the value in the keystore
func keystoreSet(key string, value PublicKeyType) error {
	keystoreShard := getKeystoreShard()
	_, present := keystoreShard[key]
	if present {
		return errors.New("entry in keystore has been taken")
	}

	keystoreShard[key] = value
	return nil
}

var KeystoreSet = keystoreSet

// Returns the value if it exists
func keystoreGet(key string) (value PublicKeyType, ok bool) {
	keystoreShard := getKeystoreShard()
```

```
206         value, ok = keystoreShard[key]
207         return
208 }
209
210 var KeystoreGet = keystoreGet
211
212 // Use this in testing to get the underlying map if you want
213 // to play with the datastore.
214 func DatastoreGetMap() map[UUID][]byte {
215         datastoreShard := getDatastoreShard()
216         return datastoreShard
217 }
218
219 // Use this in testing to get the underlying map if you want
220 // to play with the keystore.
221 func KeystoreGetMap() map[string]PublicKeyType {
222         keystoreShard := getKeystoreShard()
223         return keystoreShard
224 }
225
226 /*
227 ********************************************
228 **        Random Byte Generator        ***
229 ********************************************
230
231 This method may help with random byte generation.
232 */
233
234 // RandomBytes. Helper function: Returns a byte slice of the specified
235 // size filled with random data
236 func randomBytes(size int) (data []byte) {
237         data = make([]byte, size)
238         _, err := rand.Read(data)
239         if err != nil {
240                 panic(err)
241         }
242         return
243 }
244
245 var RandomBytes = randomBytes
246
247 /*
248 ********************************************
249 **            KDF                  **
250 **         Argon2Key                **
251 ********************************************
252  */
253
254 // Argon2:  Automatically chooses a decent combination of iterations and memory
255 // Use this to generate a key from a password
256 func argon2Key(password []byte, salt []byte, keyLen uint32) []byte {
257         result := argon2.IDKey(password, salt, 1, 64*1024, 4, keyLen)
```

```
258         return result
259 }
260
261 var Argon2Key = argon2Key
262
263 /*
264 ******************************************
265 **            Hash                  **
266 **           SHA512                   **
267 ******************************************
268  */
269
270 // SHA512: Returns the checksum of data.
271 func hash(data []byte) []byte {
272         hashVal := sha512.Sum512(data)
273         // Converting from [64]byte array to []byte slice
274         result := hashVal[:]
275         return result
276 }
277
278 // Hash returns a byte slice containing the SHA512 hash of the given byte slice.
279 var Hash = hash
280
281 /*
282 ******************************************
283 **        Public Key Encryption         **
284 **      PKEKeyGen, PKEEnc, PKEDec       **
285 ******************************************
286  */
287
288 // Four structs to help you manage your different keys
289 // You should only have 1 of each struct
290 // keyType should be either:
291 //     "PKE": encryption
292 //     "DS": authentication and integrity
293
294 type PKEEncKey = PublicKeyType
295 type PKEDecKey = PrivateKeyType
296
297 type DSSignKey = PrivateKeyType
298 type DSVerifyKey = PublicKeyType
299
300 // Generates a key pair for public-key encryption via RSA
301 func pkeKeyGen() (PKEEncKey, PKEDecKey, error) {
302         RSAPrivKey, err := rsa.GenerateKey(rand.Reader, rsaKeySizeBits)
303         RSAPubKey := RSAPrivKey.PublicKey
304
305         var PKEEncKeyRes PKEEncKey
306         PKEEncKeyRes.KeyType = "PKE"
307         PKEEncKeyRes.PubKey = RSAPubKey
308
309         var PKEDecKeyRes PKEDecKey
```

```
	PKEDecKeyRes.KeyType = "PKE"
	PKEDecKeyRes.PrivKey = *RSAPrivKey

	return PKEEncKeyRes, PKEDecKeyRes, err
}

var PKEKeyGen = pkeKeyGen

// Encrypts a byte stream via RSA-OAEP with sha512 as hash
func pkeEnc(ek PKEEncKey, plaintext []byte) ([]byte, error) {
	RSAPubKey := &ek.PubKey

	if ek.KeyType != "PKE" {
		return nil, errors.New("using a non-pke key for pke")
	}

	ciphertext, err := rsa.EncryptOAEP(sha512.New(), rand.Reader, RSAPubKey, plaintext, nil)

	if err != nil {
		return nil, err
	}

	return ciphertext, nil
}

var PKEEnc = pkeEnc

// Decrypts a byte stream encrypted with RSA-OAEP/sha512
func pkeDec(dk PKEDecKey, ciphertext []byte) ([]byte, error) {
	RSAPrivKey := &dk.PrivKey

	if dk.KeyType != "PKE" {
		return nil, errors.New("using a non-pke for pke")
	}

	decryption, err := rsa.DecryptOAEP(sha512.New(), rand.Reader, RSAPrivKey, ciphertext, nil)
	if err != nil {
		return nil, err
	}

	return decryption, nil
}

var PKEDec = pkeDec

/*
*****************************************
**         Digital Signature           **
**      DSKeyGen, DSSign, DSVerify     **
*****************************************
 */

```

```
// Generates a key pair for digital signature via RSA
func dsKeyGen() (DSSignKey, DSVerifyKey, error) {
	RSAPrivKey, err := rsa.GenerateKey(rand.Reader, rsaKeySizeBits)
	RSAPubKey := RSAPrivKey.PublicKey

	var DSSignKeyRes DSSignKey
	DSSignKeyRes.KeyType = "DS"
	DSSignKeyRes.PrivKey = *RSAPrivKey

	var DSVerifyKeyRes DSVerifyKey
	DSVerifyKeyRes.KeyType = "DS"
	DSVerifyKeyRes.PubKey = RSAPubKey

	return DSSignKeyRes, DSVerifyKeyRes, err
}

var DSKeyGen = dsKeyGen

// Signs a byte stream via SHA256 and PKCS1v15
func dsSign(sk DSSignKey, msg []byte) ([]byte, error) {
	RSAPrivKey := &sk.PrivKey

	if sk.KeyType != "DS" {
		return nil, errors.New("using a non-ds key for ds")
	}

	hashed := sha512.Sum512(msg)

	sig, err := rsa.SignPKCS1v15(rand.Reader, RSAPrivKey, crypto.SHA512, hashed[:])
	if err != nil {
		return nil, err
	}

	return sig, nil
}

var DSSign = dsSign

// Verifies a signature signed with SHA256 and PKCS1v15
func dsVerify(vk DSVerifyKey, msg []byte, sig []byte) error {
	RSAPubKey := &vk.PubKey

	if vk.KeyType != "DS" {
		return errors.New("using a non-ds key for ds")
	}

	hashed := sha512.Sum512(msg)

	err := rsa.VerifyPKCS1v15(RSAPubKey, crypto.SHA512, hashed[:], sig)

	if err != nil {
		return err
```

```
414         } else {
415                 return nil
416         }
417 }
418
419 var DSVerify = dsVerify
420
421 /*
422 ********************************************
423 **              HMAC                  **
424 **        HMACEval, HMACEqual         **
425 ********************************************
426  */
427
428 // Evaluate the HMAC using sha512
429 func hmacEval(key []byte, msg []byte) ([]byte, error) {
430         if len(key) != 16 { // && len(key) != 24 && len(key) != 32 {
431                 return nil, errors.New("input as key for hmac should be a 16-byte key")
432         }
433
434         mac := hmac.New(sha512.New, key)
435         mac.Write(msg)
436         res := mac.Sum(nil)
437
438         return res, nil
439 }
440
441 var HMACEval = hmacEval
442
443 // Equals comparison for hashes/MACs
444 // Does NOT leak timing.
445 func hmacEqual(a []byte, b []byte) bool {
446         return hmac.Equal(a, b)
447 }
448
449 var HMACEqual = hmacEqual
450
451 /*
452 ********************************************
453 **   Hash-Based Key Derivation Function   **
454 **              HashKDF                 **
455 ********************************************
456  */
457
458 // HashKDF (uses the same algorithm as hmacEval, wrapped to provide a useful
459 // error)
460 func hashKDF(key []byte, msg []byte) ([]byte, error) {
461         if len(key) != 16 {
462                 return nil, errors.New("input as key for HashKDF should be a 16-byte key")
463         }
464
465         mac := hmac.New(sha512.New, key)
```

```
466         mac.Write(msg)
467         res := mac.Sum(nil)
468 
469         return res, nil
470 }
471 
472 var HashKDF = hashKDF
473 
474 /*
475 ********************************************
476 **       Symmetric Encryption           **
477 **         SymEnc, SymDec               **
478 ********************************************
479  */
480 
481 // Encrypts a byte slice with AES-CTR
482 // Length of iv should be == AESBlockSizeBytes
483 func symEnc(key []byte, iv []byte, plaintext []byte) []byte {
484         if len(iv) != AESBlockSizeBytes {
485                 panic("IV length not equal to AESBlockSizeBytes")
486         }
487 
488         block, err := aes.NewCipher(key)
489         if err != nil {
490                 panic(err)
491         }
492 
493         // The IV needs to be unique, but not secret. Therefore it's common to
494         // include it at the beginning of the ciphertext.
495         ciphertext := make([]byte, AESBlockSizeBytes+len(plaintext))
496 
497         mode := cipher.NewCTR(block, iv)
498         mode.XORKeyStream(ciphertext[AESBlockSizeBytes:], plaintext)
499         copy(ciphertext[:AESBlockSizeBytes], iv)
500 
501         return ciphertext
502 }
503 
504 var SymEnc = symEnc
505 
506 // Decrypts a ciphertext encrypted with AES-CTR
507 func symDec(key []byte, ciphertext []byte) []byte {
508         block, err := aes.NewCipher(key)
509         if err != nil {
510                 panic(err)
511         }
512 
513         if len(ciphertext) < AESBlockSizeBytes {
514                 panic("ciphertext too short")
515         }
516 
517         iv := ciphertext[:AESBlockSizeBytes]
```

```
518         ciphertext = ciphertext[AESBlockSizeBytes:]
519
520         plaintext := make([]byte, len(ciphertext))
521
522         mode := cipher.NewCTR(block, iv)
523         mode.XORKeyStream(plaintext, ciphertext)
524
525         return plaintext
526 }
527
528 var SymDec = symDec
529
530 // If DebugOutput is set to false, then DebugMsg will suppress output.
531 var DebugOutput = true
532
533 // Feel free to use userlib.DebugMsg(...) to print strings to the console.
534 func DebugMsg(format string, args ...interface{}) {
535         if DebugOutput {
536                 msg := fmt.Sprintf("%v ", time.Now().Format("15:04:05.00000"))
537                 log.Printf(msg+strings.Trim(format, "\r\n ")+"\n", args...)
538         }
539 }
540
541 // Deterministically converts a byte slice to a string of length 128 that is
542 // suitable to use as the storage key in a map and marshal/unmarshal to/from
543 // JSON.
544 func MapKeyFromBytes(data []byte) (truncated string) {
545         return fmt.Sprintf("%x", sha512.Sum512(data))
546 }
547
```