Levi Amen
Quentin Covert
Mark Hernandez
Cameron Johnson
Collin Victor

University of Nebraska Lincoln

CSCE 361
Dr. Cohen

Project Specifications

1. Introduction

Imagine for a moment that you're an undercover spy tucked away in a faraway nation. Your mission is simple, to perform reconnaissance on the military aspirations of the local government. Every successful spy mission starts the same way, research. You prepared a set of dossiers about each bureaucrat you are likely to meet, and now you need a way to access them in the field. Some would recommend using dropbox or google drive to sync your files between your base of operations and your field assignment. However, beware! If you upload sensitive documents to existing syncing solutions, your dossiers are available, unencrypted, on those servers. They could be accessed through hacking, or viewed directly by the syncing service itself. This group project sets out to solve this issue, by creating a remote syncing application in which security and transparency is a top priority.

1.1. Purpose

Most existing file syncing applications are not as secure as they could be. The files may be uploaded over an encrypted https or sftp connection, but once the remote server receives them, they are either stored unencrypted or are available in some form to the maintainers of the service. This liability opens the files to potential access from hackers or from employees of the service provider.

To resolve this issue, SyncOrSwim implements a file syncing system using end-to-end encryption. In this scheme, the files stored on the server are encrypted using a symmetric encryption algorithm. Symmetric encryption means that the same key used to encrypt the file is necessary to decrypt it. With SyncOrSwim, this file is stored locally on the client's computer. Thus, any attempts to access the files on the server would require access to the client's computer. The key is never transmitted over the network, and is completely controlled by the end-user. Combining all of these properties creates an incredibly secure and useful service.

1.2. Document Conventions

To differentiate encryption the program performs during syncing to remote file system, and encryption performed manually by the user, all references of encryption performed by the program during the syncing process will be defined as "auto-encrypt" in this document.

1.3. Intended Audience and Reading

This document is intended to be read by developers of the product and consumers of the final product. This product is built for individual users who would like to have additional security and privacy of their files.

1.4. Product Scope

SyncOrSwim is designed as a product for use by a single, individual person. The product has no plans for group features, or version history tracking available in solutions like git or subversion.

The product considers all aspects of syncing files between a local directory and a remote location to be within its scope.

Similarly, SyncOrSwim will encompass a few cryptography functions. These include encrypting or decrypting files using a symmetric encryption algorithm, as well as the generation of cryptographic keys for use in these functions. Keeping these keys is left to the end-user, recovery of lost keys and copying keys between machines is not within the scope of the product.

1.5. References

Twitter's watchman git repository:

https://github.com/twitter/watchman

Watchman's documentation:

https://facebook.github.io/watchman/docs/install.html

The project repository:

GitHub: https://github.com/QuentinCovert/SyncOrSwim

GitLab: https://git.unl.edu/cjohnson86/CSCE361product

2. Overall Description

2.1. Product Perspective

SyncOrSwim will provide secure file syncing functionality. Clients will be able to interact with a graphical user interface (GUI) to specify files they wish to be secured with encryption and uploaded to a remote directory. The system can be configured to watch one or more root directories and their subdirectories. To the end user, the directories will appear as normal, but the remote will store only encrypted files.

It is optional for the system to not encrypt files on the remote file structure. Also, it is optional for not all files in the target root directories to be present on the remote.

2.2. Product Functions

2.2.1. General Requirements

1. The program shall watch one or more root directory trees.
2. The program shall allow the user to specify files and directories, inside root directories, to be ignored by the system.

3. The program shall allow the user to specify a remote filesystem for files to be synced to.

4. The program shall automatically sync changed files, under root directories, to remote filesystem.

5. The program shall provide the user the ability to choose how files are synced to remote filesystem.

    a. The program shall encrypt files and sync to remote or,

    b. The program shall not encrypt files and sync to remote.

6. The program shall provide to the user a record of each file, watched by the program, containing:

    a. The file's path relative to its root directory.

    b. Time of latest modification of file.

    c. Time of latest sync of the file.

    d. If encryption is enabled for the file.

7. The program shall update local files if files on the remote filesystem have a more recent modification time.

2.2.2. Crypto Requirements

1. The crypto module shall provide a method to generate a new 32-bit symmetric key in the fernet format.

2. The crypto module shall automatically produce a 32-bit symmetric, fernet format, key for the user if none is provided.

3. The crypto module shall provide a method to encrypt target files.

    a. This method will be automatically used by the program.

    b. This method can be manually used by the user.

4. The crypto module shall provide a method to decrypt target files.

    a. This method will be automatically used by the program.

    b. This method can be manually used by the user.

5. The crypto module shall automatically encrypt synced files below a specified size.
    a. The default size for the crypto module shall be 2GB.
    b. The crypto module shall provide a method for the user to specify a custom max size.

2.2.3. GUI Requirements

1. The program shall present the user with a GUI once the user runs the program executable.
2. The GUI shall provide means for the user to exit.
3. The GUI shall provide means for root directories to be added the the program's watch list.
4. The GUI shall provide means for root directories to be removed from the program's watch list.
5. The GUI shall provide means for files or directories, under watched root directories, to not be automatically synced (ignored).
6. The GUI shall provide means for auto-encryption to be enabled for files or directories, under watched root directories.
7. The GUI shall provide means for auto-encryption to be disabled for files or directories, under watched root directories.
8. The GUI shall display to the user relevant information of a selected file, contained in watched root directories. See 2.2.1 part 6 for specified information.
9. The GUI shall provide means for max auto-encryption files size to be adjusted by the user.
10. The GUI shall provide means for the user to specify a target remote file system, for the program to sync to.
11. The GUI shall provide means for local files to be manually encrypted. These files do not need to be under root directories watched by the program.
12. The GUI shall provide means for local files to be manually unencrypted. These files do not need to be under root directories watched by the program.

13. The GUI shall provide means for the user to set a given 32-bit symmetric, fernet format key as the encryption key used by the program.
14. The GUI shall provide mean for generation of 32-bit symmetric, fernet format keys.

2.3. User Classes and Characteristic

General User: The general user of this system may use the GUI to setup automated encrypting and syncing of local files with a remote filesystem. The user may set up a list of root directories to be watched by the system. When the user saves files and folders into the root directories they will be copied to the remote file system securely.

Security Wary Users: Users that do not wish to sync files but simply want to encrypt and decrypt local files may use this product to accomplish these tasks. The GUI interface will provide these users with the ability to encrypt and decrypt files locally on their machine.

2.4. Operating Environment

This product is dependent on Watchman and is cross platform - available on Windows, Mac, and Linux. To ensure the open nature of Watchman, our product is coded in Python - making it compatible with all major operating systems. Most functions of the program will require an internet connection.

2.5. Design and Implementation Constraints

1. The program is constrained to a single user. It does not support group file management, and is not a replacement for proper group file management solutions like git and subversion.
2. Due to the resource demand of encrypting large files, encryptable file sizes will be constrained to a maximum value. By default this value will be 2 gigabytes. Any files over the maximum may be synced, but not automatically encrypted. The system will still allow manual encrypting of large files, with the understanding that the task may take a significant amount of time.
3. In order to better use system memory, files over a particular size will be encrypted by splitting the file into blocks of 10 megabytes. That is, instead of loading the entire file into memory, the code will load and encrypt 10 megabytes sections at a time. This size will default to 400 Megabytes.
4. The program shall obfuscate files in the remote directory. This shall be attained by naming encrypted files a random string, and by placing all files in the same directory.
5. The system shall maintain an SQLite database that tracks all watched files. This database will store a local path for the unencrypted file, a path for the encrypted file's

temporary storage on sync, a record of the time of the file's last modification, and the status of the file's encryption.

6. Due to the non-scalability of remote servers, each user will have a maximum amount of space in which to store their files remotely. This storage space shall be 15 gigabytes.

2.6. Assumptions and Dependencies

1. This program makes several assumptions about the amount of space the user has available. This program assumes that the user has enough available space to store all of the files that are downloaded from the remote. It is also assumed that, in the case of a local remote, the remote has enough space to store all of the files that are synced with it.
2. This program is dependent on Watchman working correctly. As such, anything that causes Watchman to stop working or work incorrectly will cause this program to fail in some cases.

3. External Hardware Requirements

3.1. User Interfaces

A GUI will be the sole primary interface with the user. This GUI will feature three key areas of interest to the user: a menu bar, a directory view and a information table. A mock up version of this GUI can be seen in Appendix A.

The menu bar holds 3 different drop downs containing functionality pertinent to the user:

- File:
    - Local file encryption.
        - Encrypt local file.
        - Decrypt local file.
    - Manage root directories.
        - Add root directory.
        - Remove root directory.
    - Create new encryption key.
    - Exit.
- Settings
    - Set target remote file system.
    - Set encryption options.
        - Set encryption key.
        - Set max auto-encrypt size.
- Help
    - Documentation.

- Contact Us.

The directory view will display graphically a list of root directories watched by the program. Each root directory will be presented as a tree view, so the user can expand or compress the directory tree structure to view files and directories contained under the root. The user can click on a file or directory in the directory view to display information regarding the file or directory in the information table.

The information table displays to the user the latest modification time, latest sync time, path relative to the root directory, encrypt enable and sync enable for a selected file, in the directory view. To change the encrypt and sync enable settings of the selected file or directory, the user can click the field to toggle the setting.

3.2. Hardware Interfaces

There are no specific hardware interfaces used expressly by this product. All operations partaken by the application are abstracted from the hardware level and only interact with other software components.

3.3. Software Interfaces

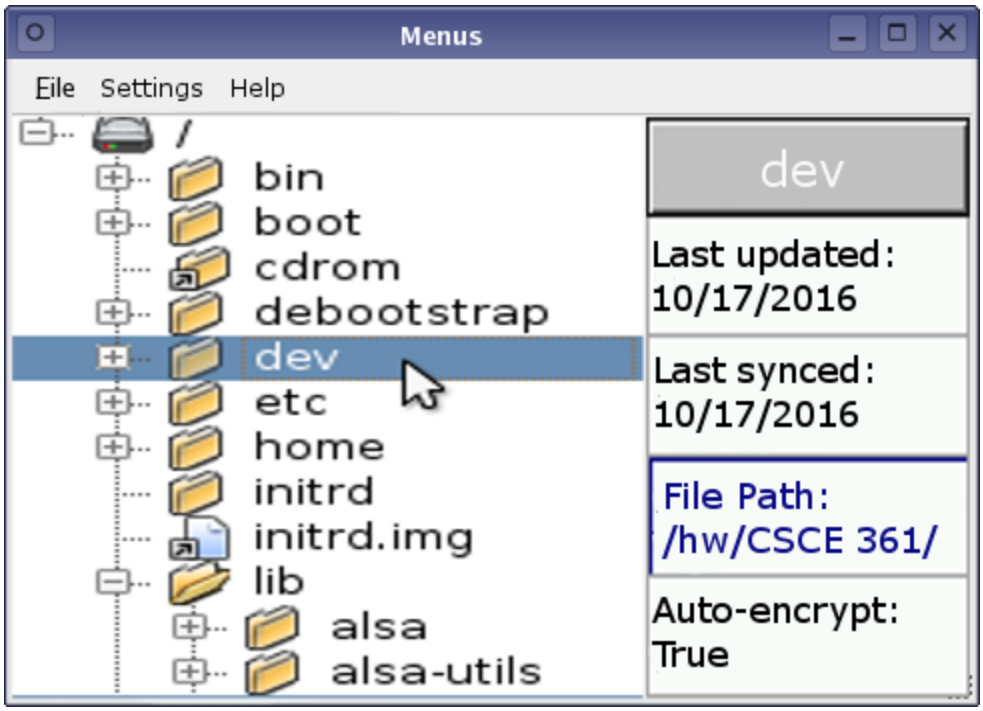
The product interfaces with Watchman via a socket interface using JSON messages. The operation system, UNIX/Linux, Mac, or Windows, provides standard filesystem interfaces for the product to interact with directory trees watched by the program. The Python library, cryptography, is used in the Crypto module to allow the program to encrypt and decrypt files. PyQt is used to create and display the GUI. SQLAlchemy is another addition that assists with the creation and maintenance of the SQLite database used by the product.

3.4. Communication Interfaces

The GUI is the primary communication source between the user and the product. Though optional, the product also makes use of an internet connection interface to allow it to sync data to/from an FTP server. To communicate with Watchman, the system uses a unix socket interface. These sockets are supported natively in python, making the interface simple to interact with.

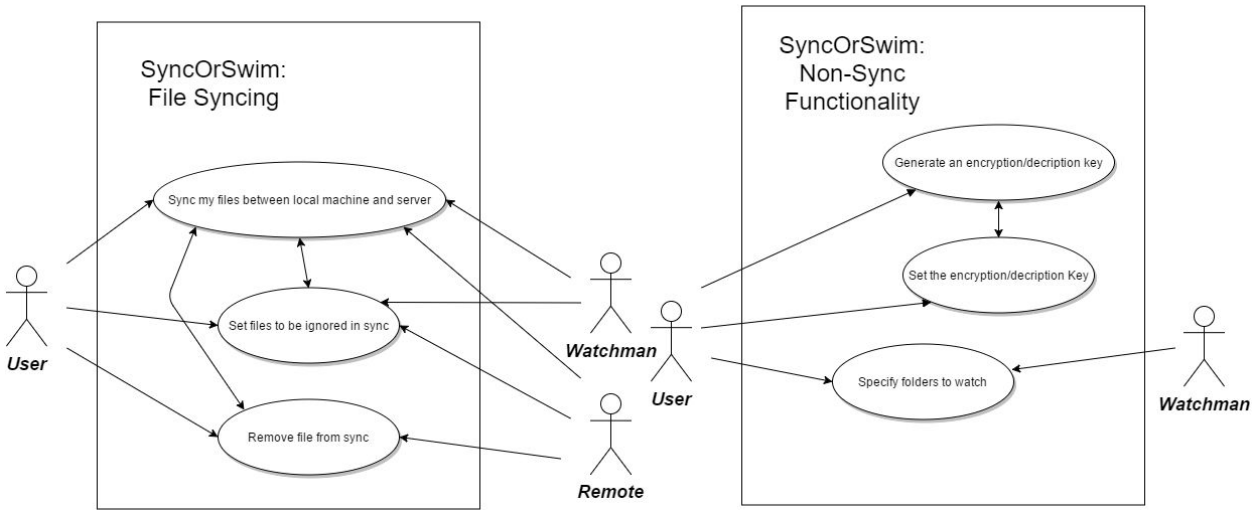
4. Appendix

Appendix A: Mock GUI interface



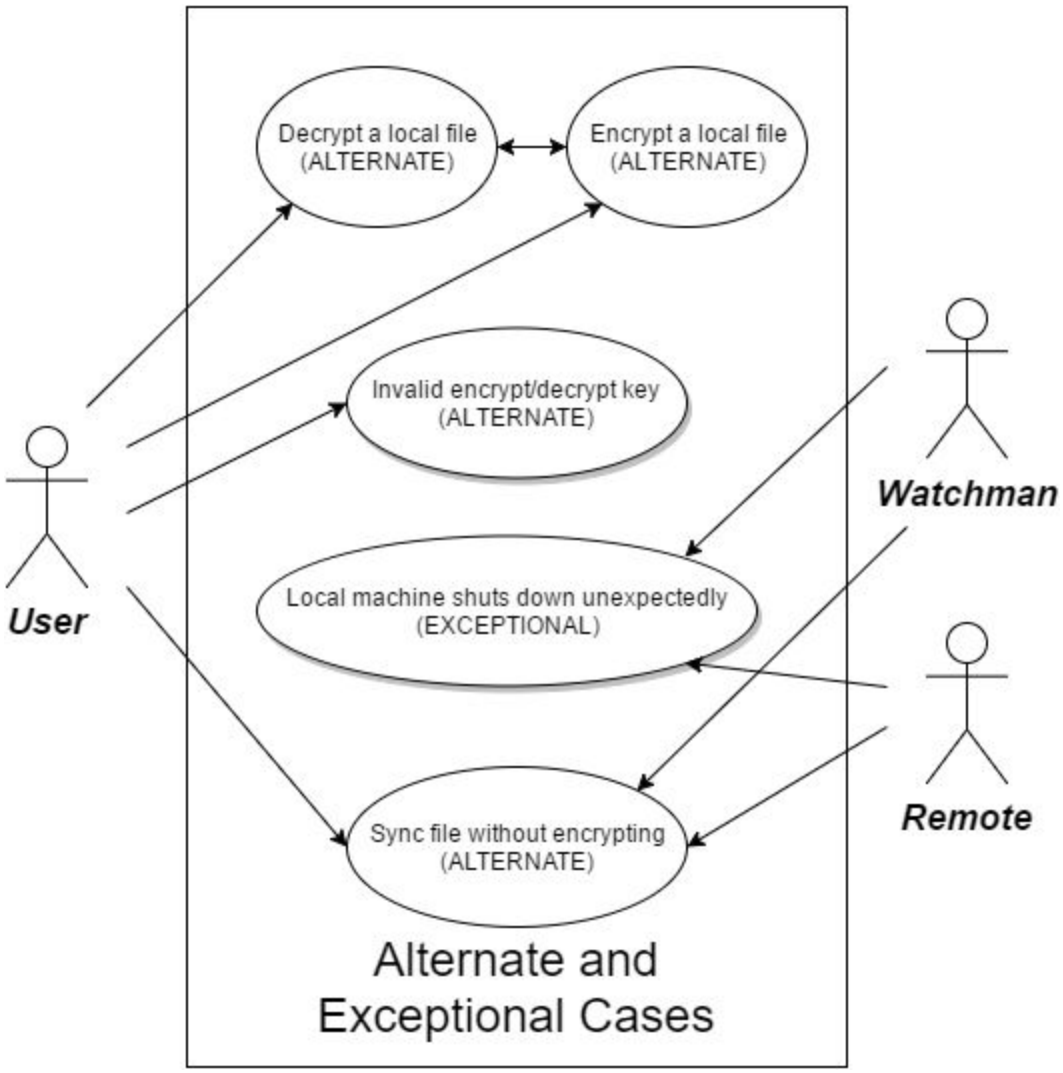
Appendix B: Annotated Use Case Diagrams

Normal Use Case Diagram



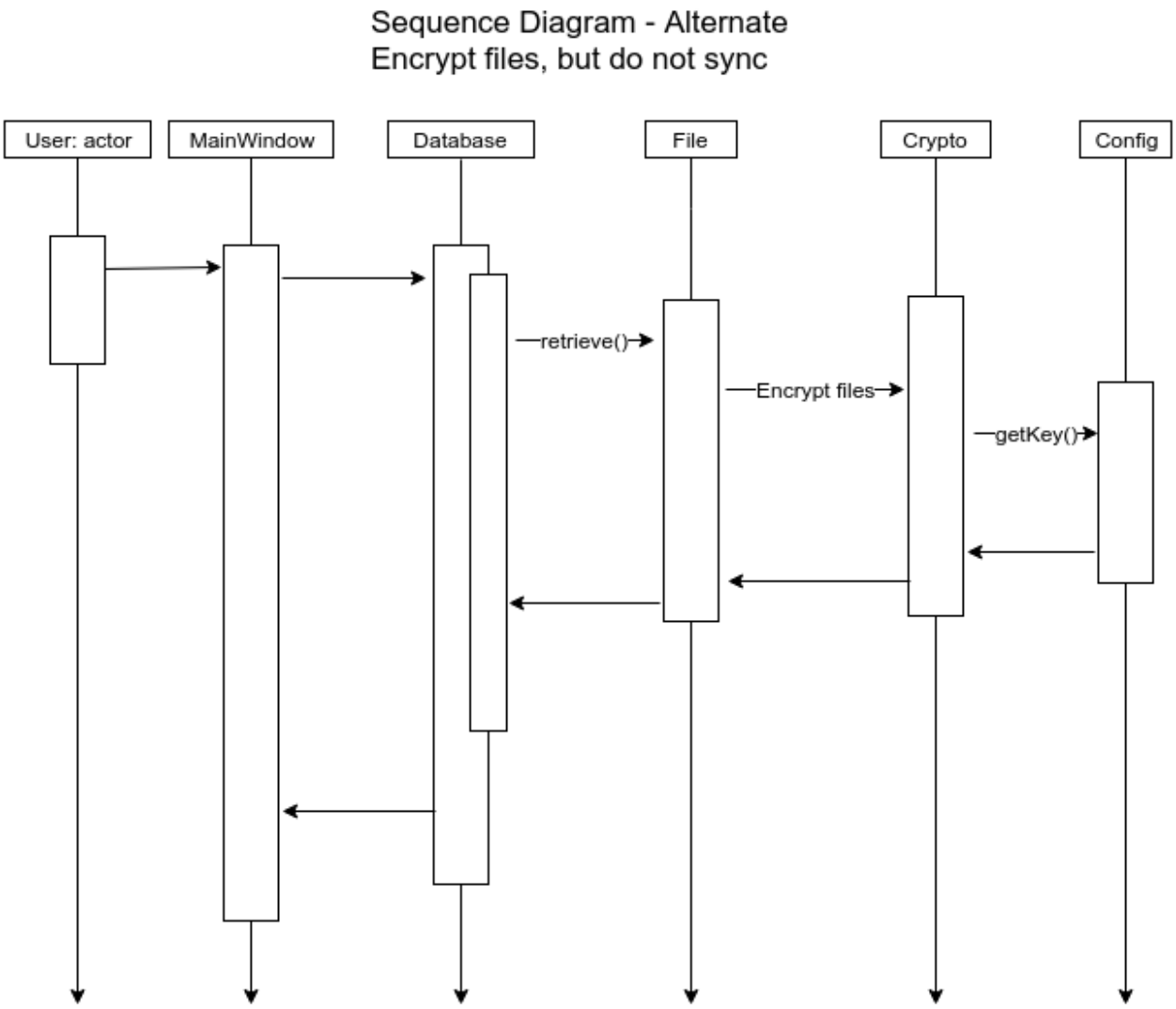
- Normal
    - File Syncing
        - Sync my files between local machine and server - Standard primary functionality of system. Takes a set of files, that is set to be encrypted by default, and syncs them between the remote server, Watchman and the local machine.
        - Set files to be ignored in sync - Keeps files from being added to set of synced files. Connected to sync files because it affects the set of files that are syncs
        - Remove file from sync - Self explanatory. Connected to sync files because it affects the set of files that are synced.
    - Non Syncing
        - Generate crypto key - Create a cryptography key for use with a particular file set
        - Set crypto key - Provide the cryptography key use with a particular file set; connected to Generate key because when you generate a key it is set as the key for the set of files related to the key
        - Specify folders to watch - Add files to Watchman's set of watched files

Alternate and Exceptional Use Case Diagram

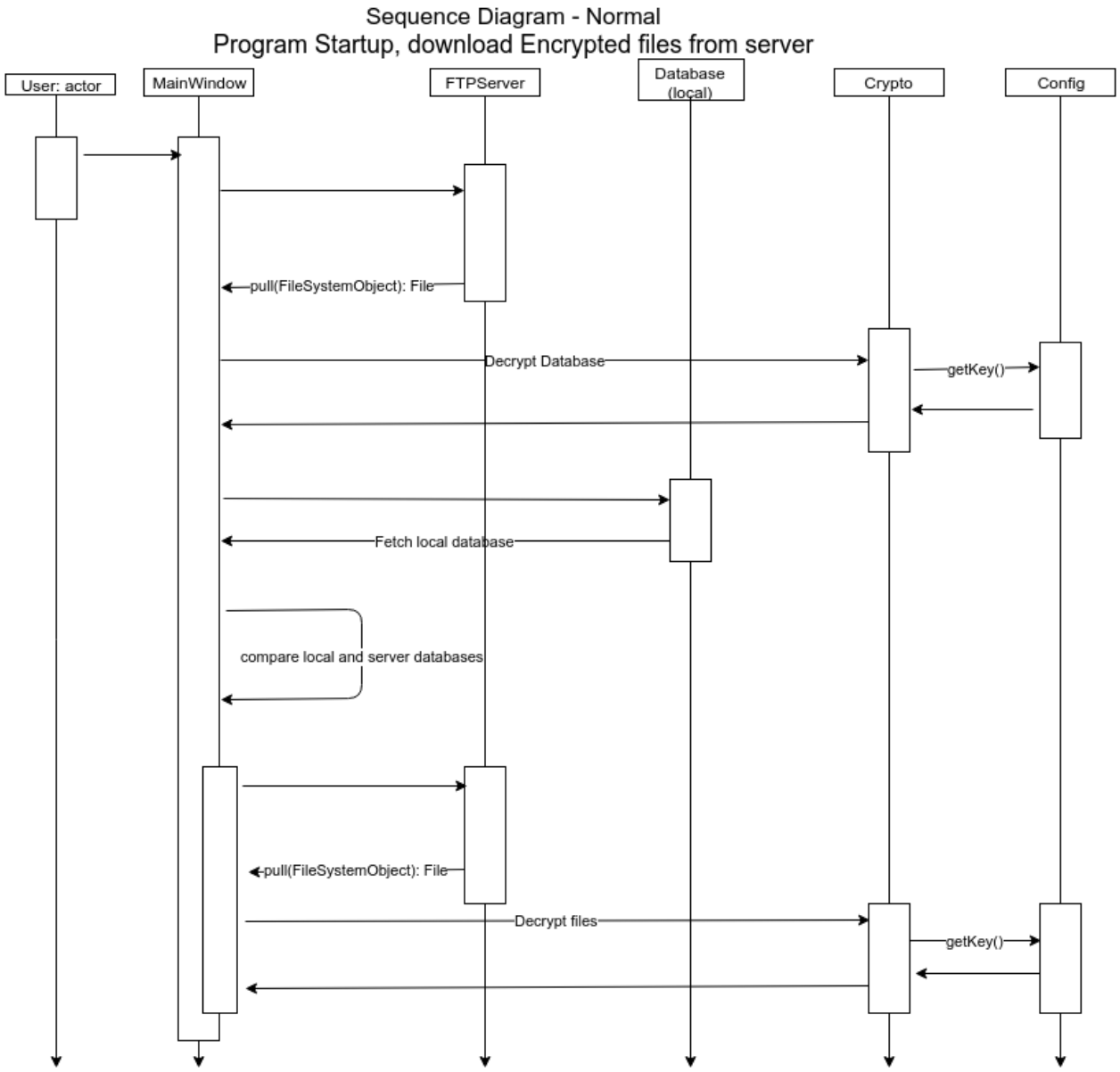


- Alternate
    - Sync file without encrypting - Sets a file in the sync file set to not be encrypted when syncing occurs.
    - Invalid crypto key - Key can not possibly used in encryption or decryption (Invalid format, invalid characters, etc.)
    - Decrypt a file - Use a key to decrypt a file/file set; this is alternative because it must be done without syncing.
    - Encrypt a file - Use a key to encrypt a file/file set; this is alternative because it must be done without syncing.
- Exceptional
    - Local Machine shuts down unexpectedly - Server-side file integrity must be maintainable despite the possibility of power failure.

Appendix C: Sequence Diagrams



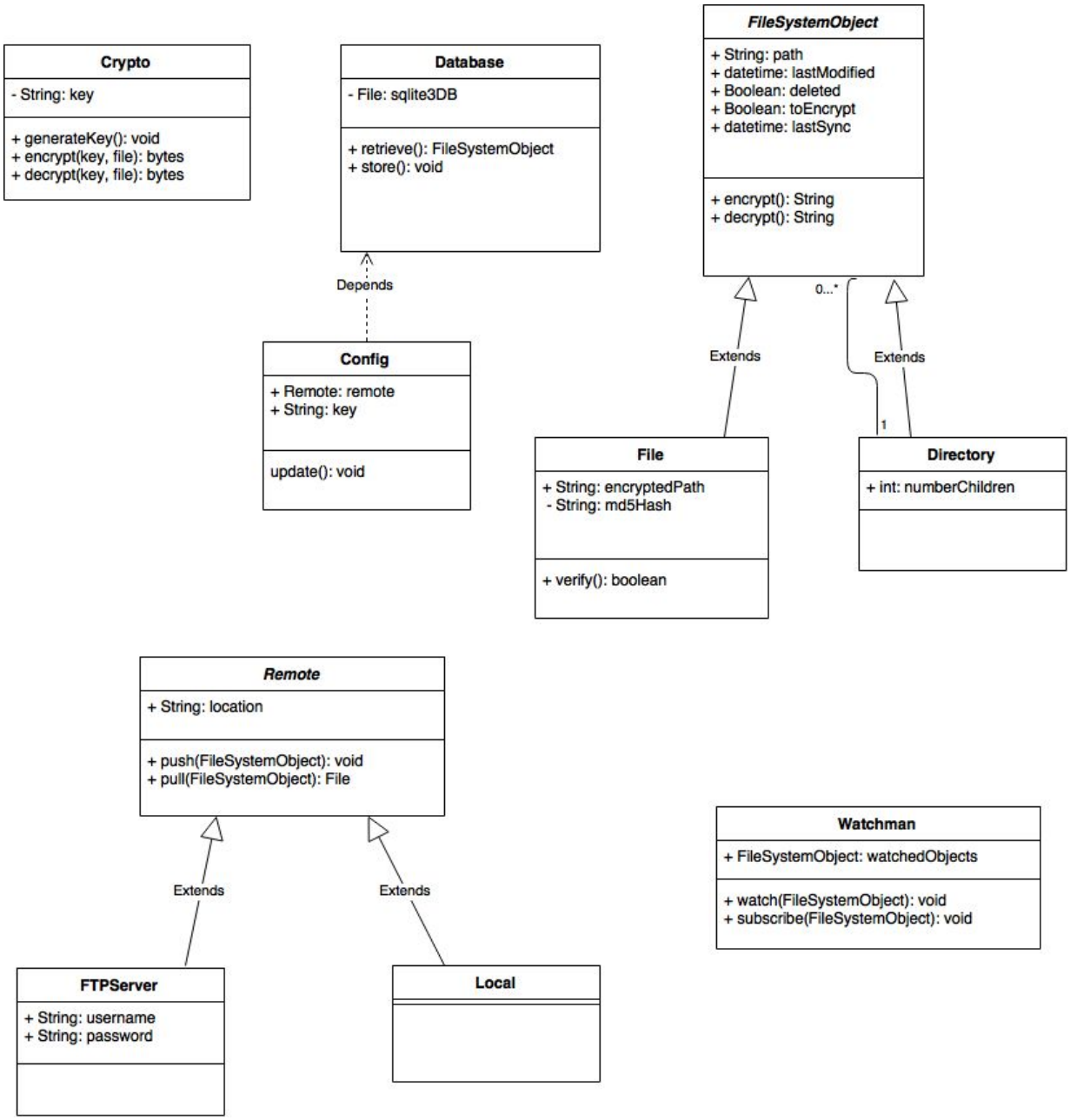
This sequence diagram is for an alternate use case, when the user requests that several files be encrypted, but not synced with a remote.



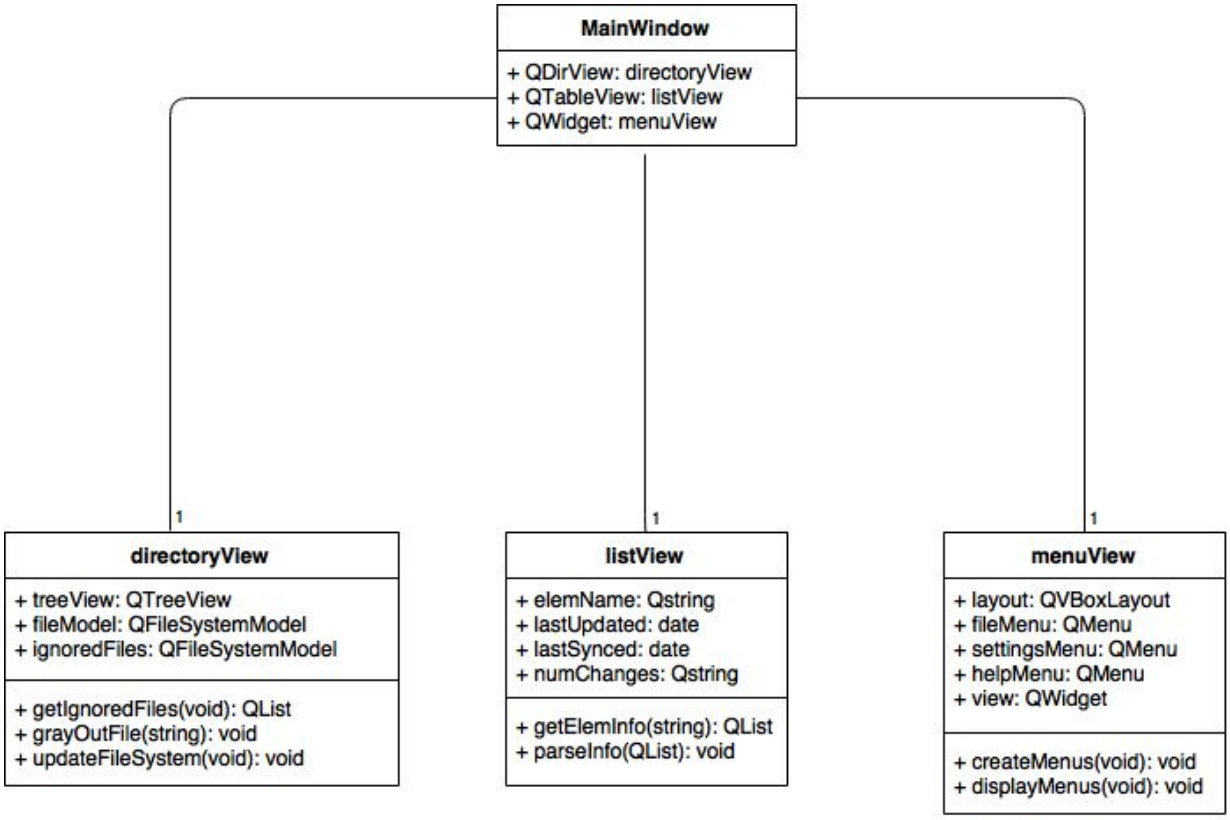
This sequence diagram is for a normal use case where, upon startup the program compares both the local and remote databases, finds that the remote database is more recent, and downloads and decrypts the files on the remote server.

Appendix D: Class Diagrams

Cryptography, Configuration, File and Remote Modules



GUI Module


MainWindow
+ QDirView: directoryView
+ QTableView: listView
+ QWidget: menuView
1
1
1
directoryView
+ treeView: QTreeView
+ fileModel: QFileSystemModel
+ ignoredFiles: QFileSystemModel
+ getIgnoredFiles(void): QList
+ grayOutFile(string): void
+ updateFileSystem(void): void
listView
+ elemName: Qstring
+ lastUpdated: date
+ lastSynced: date
+ numChanges: Qstring
+ getElemInfo(string): QList
+ parseInfo(QList): void
menuView
+ layout: QVBoxLayout
+ fileMenu: QMenu
+ settingsMenu: QMenu
+ helpMenu: QMenu
+ view: QWidget
+ createMenus(void): void
+ displayMenus(void): void