SimpleumSafe

Technical White Paper

Document Version: *2.0*
Document Date: *2017-11-17*

Software Version: *2.0 macOS, 2.0 iOS*

CONTENT

Principles

Motivation

Digital data are often worth to protect to ensure privacy, trust, obligations and confidentiality as well as to protect the data against crime attacks. But the application and understanding of existing encryption solutions is still too complicated for many users. Therefore, there users do not protect their data. Simpleum wants to change this.

Product Design Goals

Master goal: "Now everybody can encrypt".

SimpleumSafe is designed to provide a simple handling of file based encryption to protect personal or professional data. Additional it provides tools to organize data like tags, favorites, comments, and to support everyday workflows like importing, exporting, previewing and editing files in a secure environment.
All these goals must be achieved by the use of an easy to understand user interface. Almost no technical background is necessary to use SimpleumSafe.

Product Design Guideline

Keep it simple and secure

- Simplicity of understanding and utilization of the User Interface
- Exclusive use of proven encryption technology
- Full integration in the Apple file workflow environment like Share, Service, Pasteboard and Drag & Drop
- No decryption of files until explicit requested (editing and previewing only with encrypted files)
- All file data and meta data are encrypted
- Addition of a complete extra security level on top of the Apple OS

Principle Data Design

When a file is imported into SimpleumSafe the binary data of the file will be encrypted and stored as a file within the Safe. The filename of the encrypted data is a unique random

number (UUID) and does not contain the original filename nor other original file attribute information.
The filename and other imported file attributes are stored with additional organizational data in an Encrypted Attribute Database (see Figure 1 - Encrypted File Storage and Figure 2 - Safe Storage Format).

After the import of files into the Safe mostly attributes are changed like the file path or the tags. Therefore, the pure file data are almost unchanged. The separation between File Data and File Attributes is useful when the Safe is backed up by Apple Time Machine or other backup software. Furthermore, this separation is useful when the Safe is stored in a cloud.

Security Software Design and Process

Internal information hiding – Simpleum®Cryptor

Every software code area can access security data only up to the absolutely necessary minimal extent, not more.
The Safe has a very small encryption and key management core (Simpleum®Cryptor). Every security operation is operated in this Cryptor. Keys are not accessible from outside the Cryptor (one-way communication). Only the next layer “The Safe Management” module can communicate with some parts of the Cryptor, but even this software layer has no access to the encryption keys.
Passwords (or their hashes) are never stored. The access to the Encryption Keys is done by providing the right Safe Password. There is no possibility for SimpleumSafe or Simpleum Media GmbH employees to decrypt a Safe without the right Safe Password provided by the User owning the Safe.

Software Development Process

1. All requirements, features and changes are traceable documented with an issue tracking system.
2. All code and configuration changes are recorded with a version control system.
3. All security relevant code areas have automated unit tests.
4. All security relevant User Interface areas have automated User Interface tests.

Data and Encryption Implementation

Every file stored in the Safe consists of two main parts:

1. The encrypted binary data (Encrypted File Data)
2. The File Attributes like filename, creation date, … (Encrypted Attribute Database)

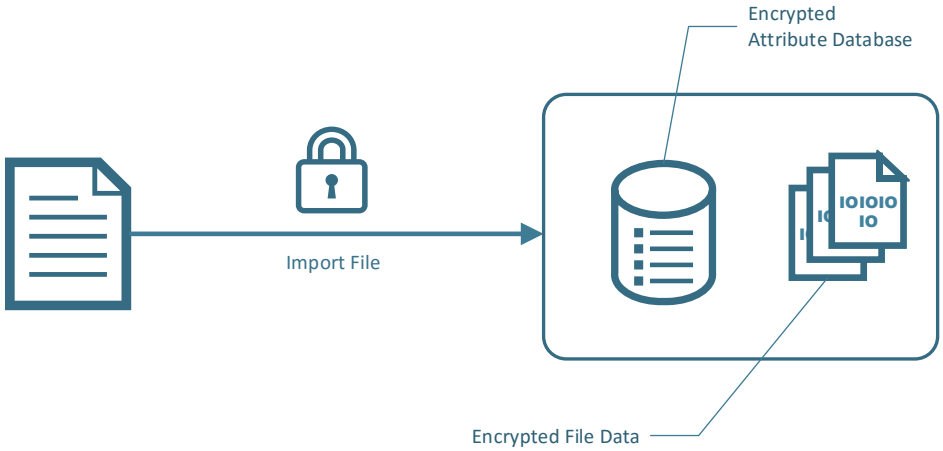

Figure 1 - Encrypted File Storage

Creation of a Safe

The user chooses a file location in a file system, a name for the Safe and a Safe Password. From the Safe Password, a symmetric encryption key is derived (described at Safe Key File). This is the Key Encryption Key (KEK) which is used to encrypt other keys (File Data Encryption Key …). The Key Encryption Key and the Safe Password are never stored. The Safe generates randomly the keys for the File Data Encryption and the Encrypted Attribute Database. Theses keys are stored in the Safe Key File encrypted with the KEK.

Open a Safe

The user chooses a Safe and enters the Safe Password. The Safe Manager asks the Simpleum®Cryptor to open the Safe by decryption the Safe Key File. There is no known information (pattern) in the Key File, so if the decryption fails, the Safe Manager cannot know if the password was wrong or the data is corrupt (by design).

Importing of Data into the Safe

Operations like import, export, backup, restore, … are done transactional in two phases. The first phase builds a work list which files should be imported and import the File Attributes in the Encrypted Attribute Database. In the second phase the File Data for every file are imported as an encrypted binary file. It is possible to pause and resume an import transaction.

Export Data from the Safe

Every file will be decrypted to the target directory creating the file hierarchy used in the Safe. Filename, creation date, … are exported too.

Backup of the Safe with Safe-Backup

The Safe copies all Encrypted File Data to the Backup target and stores for every file the File Attributes as an encrypted XML-File. This format is used to provide a simple backup file format which can easily be restored even when technology changes over time, because this format is not OS depended.

Safe Data Format

Each SimpleumSafe has the following file structure:

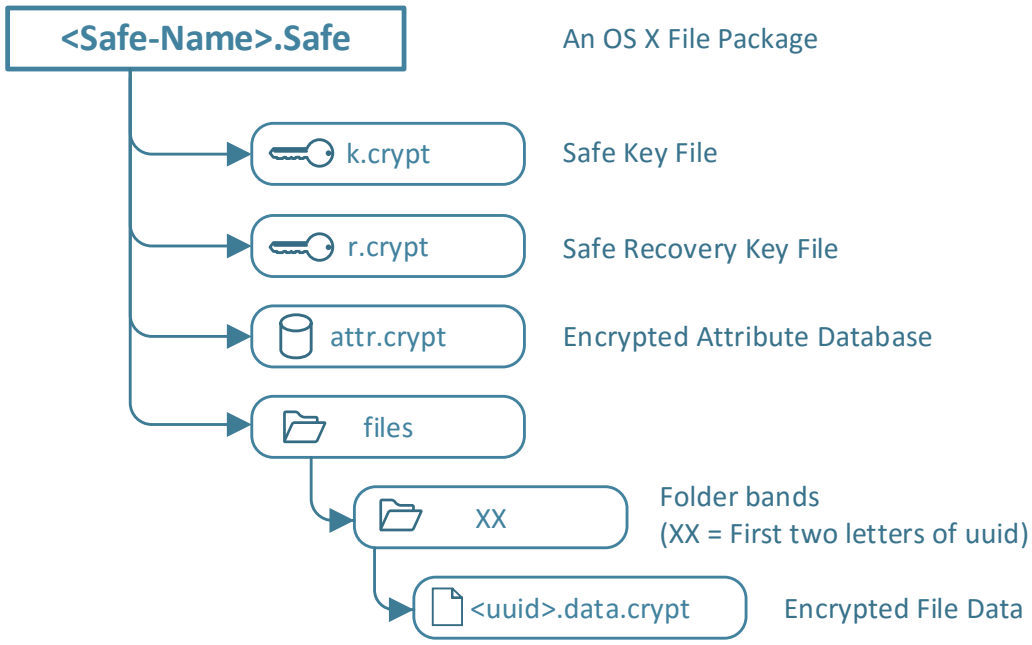

Figure 2 - Safe Storage Format

Safe Key File

The Safe Key File contains the File Data Encryption Key and the Attribute Database Encryption Key. The Safe Key File is encrypted in RNCryptor-Format V3 using the Safe Password provided by the User while creation of the safe.
To derivate strong keys from a user given password the following proven algorithm is used:

1. Generate a random Encryption Salt.
2. Generate the encryption key using PBKDF2.
    a. Pass the password as a string
    b. Pass the random encryption salt
    c. Run PBKDF2 10,000 iterations
    d. Create SHA-1 PRF. Request a length of 32 bytes.
3. Generate a random HMAC salt and HMAC-key (using PBKDF2 similar to encryption key).
4. Generate a random Initialization Vector (IV)
5. Encrypt the data using the encryption key and IV using AES-256 (CBC mode).

6. Generate message HMAC: Pass header, ciphertext, HMAC-key and SHA-256 to HMAC function to generate HMAC
7. Put these elements together in the format given as Safe Key File.

Generation of Safe Recovery Key File (optional)

The unencrypted content of the Recovery Key File is the same as for the Safe Key File. It is also a password encrypted. The only difference between the two key files is that it uses a randomly generated password, which can be generated on user demand with the authorized Safe Password. The generated password is shown to the user who should write it down or to print it. The user should store this Recovery Password at a safe place. This Recovery Password would be a safety net in the case the user has forgotten the Safe Password.

Encrypted Attribute Database

Each file has several File Attributes like filename, creation date, modification data, tags, comment, add to Safe date and trash status. The data of these File Attributes are stored in a SQLite Database for fast and transactional access. This Database itself is encrypted by application of the widely used SQLCipher implementation.
SQLCipher provides low-level file encryption of the SQLite Database with key-based AES-256 encryption. The key for database encryption is additionally stored in the Safe Encryption Key File.

Files Directory and Encrypted File Data

Each file in the Safe has a unique UUID and its related Encrypted Data File has <UUID>.data.crypt as filename.
The directory named “files” contains all Encrypted File Data. It is divided into bands of files to minimize the number of files per directory. The Directory names consist of the first two letter of the containing UUIDs. The Apple File system HPF+ can contain 2.1 billion files in one directory.
The Encrypted File Data files are encrypted in RNCryptor-Format V3 with AES-256.

Secure Temporary Storage (only macOS)

For some reasons, it is necessary to access files from the Mac file system (for example when a file should be edited by an external application or when a file is previewed).
SimpleumSafe utilizes a macOS encrypted Sparse Image with AES-256 encryption.

Mount

1. Copy an empty encrypted sparse image to the application data directory
2. Generate a random password
3. Change the sparse image password into the new generated password
4. Mount the Sparse Image

Unmount

1. Unmount the sparse Image
2. Generate a random password for one-time usage
3. Change the sparse image password into the new generated password
4. Password is not stored

There is no easy and fast way to securely erase data on SSD devices, which are mostly used by modern Apple Macs. Therefore, the change of the Sparse Image password and "throw it away" makes the data unusable and is fast. The Sparse Image will never be used again after it was unmounted.
At every start of SimpleumSafe a new Encrypted Temporary Storage will be created.

Usage of the Secure Temporary Storage

When the User wants to edit a file of the Safe, the file will be exported into the Secure Temporary Storage and the related App will be started with the file as an argument. The started App and the exported file are observed.
When a "file save event" is recognized, then the edited file will be immediately imported. After closure of the called App the exported file will be deleted in the Secure Temporary Storage.

Prevention against Unauthorized Change of Security Settings

If the user wants to change a security relevant setting, like Safe Password or automatic Safe lock settings, an authorization with the correct Safe Password is necessary again.

Prevention against User Interface Brute Force Attacks

If an entered password for authorization was wrong, the next possibility to enter a password again is delayed. The first delay is 2 seconds, then 4, 8, 16, 32 seconds, and so on.

Password Strength Analysis

SimpleumSafe will give the user a hint about the password strength every time when a new password is entered. There is no algorithm available which can calculate the time needed to crack a password. Therefore, SimpleumSafe implemented the "Zxcvbn" algorithm developed by Dropbox Inc., which can give the user an idea about the quality of the entered password.

Data Synchronization

Beginning with SimpleumSafe 2.0 a synchronization of Safes between different devices is possible.

Synchronization Design Goals

- Fully encrypted on local device and on cloud storage/Sync Target.
- Protect even if data was stolen from Cloud
- No server logic required (Peer-to-Peer, file-based)
- Easy to use and easy to configure
- Tolerant against not exact same time on different devices
- Extendable to different Cloud providers
- Enable Sync without Internet (local Network or Bluetooth)
- Enable Sync without Network for extreme high security areas (no galvanic / electric connection between devices) e.g. use an USB-Stick as a sync Target

Concept and implementation

SimpleumSafe uses an asynchronously decentral synchronization. This concept is mostly known from the version control system "git".

Meta Files Sync

The abstract concept of asynchronously decentral synchronization requires that all changes in the Safe are recorded as transaction logs (insert, update, delete). Every Safe on every device has an own transaction log. In the first step of the sync, all transaction logs are exchanged between the devices. Afterwards all "new" transactions from other devices are ordered in relation to the local transactions. The ordering is done by utilizing vector clocks. This enables a correct ordering of transactions even when the timestamps of the transactions from the different devices are not the same.
As the next step these ordered transactions are integrated/replayed into the local Safe. All new transactions in the local Safe are exported and will then be available for the other devices.

Data Files Sync

For every inserted or updated entry which has a corresponding data file (files, no directories) an entry is added in the Sync File Transfer List (STFL) for up- or download. After a successful download of a data file the corresponding file entry is marked as loaded (not loaded entries are displayed as gray entries). So, the user can see the file entries even when the download of the corresponding data files hasn't finished, which can take a while, depending on the cloud service and the network connection.

Data types used for sync

- Event-Database aka transaction log: Implemented as a SQLite Database where all Safe relevant data are encrypted
- Sync folders at sync target:
    - baseline: consolidated transactions: JSON files where all Safe relevant data are encrypted

- evens: not consolidated transactions: JSON files where all Safe relevant data are encrypted
- files: these are the encrypted data files from the Safe

The Safe Key Files is not stored in the Sync Target!

Synchronization Types

iCloud Sync

The iCloud Sync utilizes Apple iCloud Drive servers. Up- and Download logic is mostly done by macOS and iOS. The Safe Sync Data is stored on servers owned by Apple or other contracted partners for hosting iCloud.

Folder Sync

The folder Sync is only available for synchronization between macOS devices. A folder (the Sync Target) must be accessible by these devices. This is mostly a NAS (Network Attached Storage) but it can although be an USB Stick which will be carried from one device to another.

Wireless Sync

Wireless Sync uses Apple Multipeer protocol which uses Bluetooth and WIFI. The devices are browsing for other devices running SimpleumSafe. The devices connect if SimpleumSafe is running on these devices with the same open Safe. The data is exchanged peer to peer without using the internet.

Synchronization Configuration

The Sync Configuration contains the Sync Type, Safe Name, Safe UUID, Safe creation date, Safe comment and the key file. The configuration is JSON encoded and encrypted with the Safe password (using RNCryptor).
For folder synchronization, an encrypted sync configuration file will be created and must be imported on the other device (Macs). To exchange the Sync Configuration between an iOS and macOS or between different iOS devices an encrypted QR code is generated and must be scanned by the other device.

No File Data Encryption Key in the cloud / Sync Target

Using this kind of exchanging the Synchronization Configuration no File Data Encryption Key is needed in the Sync Target. So, the data encryption key is no available in the cloud / Sync Target.
As a consequence: It is not possible to decrypt the Sync Data even when the Safe password is known.

Glossary

Advanced Encryption Standard (AES)

AES is a symmetric block cipher algorithm which is the best analyzed and widely used encryption algorithm. It is used by bank corporations, governments and many more institutions to protect their data. SimpleumSafe uses AES-256.

Encrypted Attribute Database

The Encrypted Attribute Database is an encrypted SQLCipher database which stores all File Attributes.

Encrypted File Data, File Data

Each file (no directory) in the Safe has an Encrypted Data File. This file contains the encrypted binary data. It does not contain File Attributes.

File

The name file is used to describe a file in the OS X, iOS file system or its representation in the Safe. Even directories are sometimes named as a file.

File Attributes

The name of a file, its location in the directory hierarchy, the creation date etc. are File Attributes. The File Attributes are stored in the Encrypted Attribute Database.

File Data Encryption Key

The Encryption Key to encrypt the File Data.

PBKDF2 (Password-Based Key Derivation Function 2)

The PBKDF2 algorithm helps to build encryption keys with a better quality and the algorithm needs additional computing time, which makes brute-force-attacks slower.

RNCryptor-Format V3

```
Byte:     |  0  |  1  |    2-9    | 10-17 | 18-33 | <-    ...    ->| n-32 - n |
Contents: | version | options | encryptionSalt | HMACSalt | IV | ... ciphertext ... |   HMAC   |
```

- version (1 byte): Data format version. Currently 3.
- options (1 byte): bit 0 - uses password
- encryptionSalt (8 bytes): if option includes "uses password"
- HMACSalt (8 bytes): if options includes "uses password"
- IV (16 bytes)
- ciphertext (variable) -- Encrypted in CBC mode

HMAC (32 bytes)
All data is in network order (big-endian).

https://github.com/RNCryptor/RNCryptor-Spec/blob/04378bc27c604e97353badbead8c435698abe97a/RNCryptor-Spec-v3.md

Safe, SimpleumSafe

A Safe is accessible and encrypted with a Safe User Password. Safes are created, opened, and accessed with the SimpleumSafe App. All files and their File Attributes are encrypted.

Safe Key File

An AES-256 encrypted file, which stores the encryption keys for the File Data and Attribute Database encryption.

Safe Password, Password

A user given password, which is needed for accessing the Safe.

Safe Recovery Key File

An optional AES-256 encrypted file, which stores the encryption keys for the File Data and Attribute Database encryption like Safe Key File, but with a randomly generated password. It is used as a secondary key a user can create and store at a safe place. It gives the user the ability to have a second key when the user forgets his Safe User Password.

SQLCipher

SQLCipher adds to the widely-used SQLite low level AES-256 file encryption of the database file. SQLCipher itself is very often used in security Apps.
https://www.zetetic.net/sqlcipher/

(Local/Remote) Sync Target

The Sync Target is a directory which contains all files needed for synchronization. To execute a Sync the device must have access to this directory. For iCloud this directory exists at least twice: on you device (this is the Local Sync Target) and on the Apple iCloud Server (Remote Sync Target). If you use "folder"-Synchronization between two Macs with a shared folder on a NAS device the local and remote Sync Target are the same.

Sync Transaction

All Operations in a Safe are recorded as inserts, updates and deletes Sync Transactions. The transactions are stored in a SQLite Database with encrypted Safe data fields.

Universally Unique Identifier (UUID)

An UUID is a 128 Bit value which is generated with the aim that it is unique and random (two generated UUID should not have the same value).

Version History

Document Version	Software Version	Date	Description
2.0	2.0	2017-11-17	Sync added
1.1	1.1.2 (macOS), 1.0 (iOS)	2016-12-20	Text corrections
1.0	1.0 (macOS)	2016-05-24	Initial version