FACULTY OF INFORMATION TECHNOLOGY CTU IN PRAGUE

ASSIGNMENT OF BACHELOR’S THESIS

Title:	Security analysis of Cryptomator
Student:	Anton Titkov
Supervisor:	Ing. Josef Kokeš
Study Programme:	Informatics
Study Branch:	Computer Security and Information technology
Department:	Department of Computer Systems
Validity:	Until the end of summer semester 2020/21

Instructions

1) Research the field of a disk and file encryption.
2) Study the tool Cryptomator (https://cryptomator.org). Describe its capabilities, compare its features and limitations to other disk or file-encryption software.
3) Analyze the user interface and available documentation of Cryptomator, evaluate these areas for threats.
4) Based on the results from the previous analysis, choose promising application components and analyze their source code for security, with a focus on cryptographic primitives.
5) Verify the documented crypto specifications by reimplementing Cryptomator's key functions using an independent cryptographic library.
6) Discuss the results.

References

Will be provided by the supervisor.

prof. Ing. Pavel Tvrdík, CSc. Head of Department	doc. RNDr. Ing. Marcel Jiřina, Ph.D. Dean

Prague February 13, 2020

Bachelor's thesis

Security analysis of Cryptomator

Anton Titkov

Department of Computer Systems
Supervisor: Ing. Josef Kokeš

June 3, 2020

Acknowledgements

I would like to thank my supervisor Ing. Josef Kokeš for his crucial help, guidance, and invaluable experience. Special thanks to Olya for being there when needed most. I would especially like to thank my family for supporting me during my studies.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular that the Czech Technical University in Prague has the right to conclude a license agreement on the utilization of this thesis as a school work under the provisions of Article 60 (1) of the Act.

In Prague on June 3, 2020 . . . .

Czech Technical University in Prague
Faculty of Information Technology
© 2020 Anton Titkov. All rights reserved.
This thesis is school work as defined by Copyright Act of the Czech Republic. It has been submitted at Czech Technical University in Prague, Faculty of Information Technology. The thesis is protected by the Copyright Act and its usage without author's permission is prohibited (with exceptions defined by the Copyright Act).

Citation of this thesis

Titkov, Anton. *Security analysis of Cryptomator*. Bachelor's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2020.

Abstract

This thesis deals with the issue of storage encryption and data safety. It provides a brief overview of technologies and software for storage encryption currently available to users.

The work provides a security analysis of Cryptomator, an open-source application for encrypting files in cloud storage. Potential risk areas are further explored and evaluated. Some of the possible attacks are shown.

Keywords cryptography, security, Cryptomator, encryption, scrypt, AES

Abstrakt

Tato práce se zabývá problematikou šifrování datového úložiště a bezpečnosti dat. Poskytuje stručný přehled technologií a softwaru pro šifrování úložiště, které jsou aktuálně dostupné uživatelům.

Práce poskytuje bezpečnostní analýzu Cryptomator, open-source aplikace pro šifrování souborů v cloudovém úložišti. Potenciální rizikové oblasti jsou dále prozkoumány a hodnoceny. Jsou uvedeny některé možné útoky.

Klíčová slova kryptografie, bezpečnost, Cryptomator, šifrování, scrypt, AES

Contents

Introduction **1**

1 Storage Security **3**
1.1 File and File System 3
1.2 Encryption technologies 5
1.3 Comparison of encryption methods 8
1.4 Existing tools for encryption 8

2 Cryptomator **11**
2.1 Features of Cryptomator 11
2.2 Architecture 11
2.3 Cryptography 16

3 Security Analysis **23**
3.1 UI Analysis 23
3.2 Source Code Analysis 26
3.3 Implementation 28
3.4 Update 1.5.0 29

Conclusion **31**

Bibliography **33**

A Acronyms **37**

B Contents of enclosed SD **39**

List of Figures

1.1 Example of a hierarchical directory system 4
1.2 Boot Sequence with FDE technology 5

2.1 KEK derivation using the scrypt KDF 13
2.2 Process of key initialization 13
2.3 Process of file/folder name encryption 14
2.4 An example of FS structure transformation 15
2.5 Process of file content encryption 16
2.6 Encryption and decryption with CTR mode 18

3.1 Settings pane 24
3.2 Vault management options 24
3.3 Password input when creating a Vault 25
3.4 Example of the masterkey replay attack 27
3.5 Cryptomator working with a vault created by the reimplementation 29

List of Tables

1.1 Comparison of encryption methods 8

2.1 Layout of the file header 15
2.2 Layout of the file chunk 16

Introduction

Today, more and more people are using cloud storage, which is becoming one of the most popular types of data storage [1]. Cloud storage is an online repository for various data (such as photos and documents) that allows access to the said data from any device. Since cloud storage providers do not store any data on the user's local storage, an internet connection is required to upload, modify, or share data. Even though privacy is a fundamental human right, most cloud storages do not provide adequate security for the data stored in the cloud [2, 3].

When an account or a cloud provider is hacked, attackers may gain access to all stored data. Government and law enforcement-issued secret surveillance programs and backdoors are also a threat. [3, 4] Anyone who has unauthorized access to an account can use the data for personal gain and wrongdoing.

Encryption helps protect the confidentiality of the data stored on computer systems or transmitted through a network.

Encryption is the process of transforming data, such as a text message or photo, into a ciphertext – data unreadable by a human without using a corresponding encryption algorithm and a valid key, so even if our computer is lost or stolen, that data is safe. When implemented properly, encryption provides close to unbreakable protection of our data.

Hence, reliable encryption methods are needed for reliable cloud storage. The issue of data encryption is continually evolving, and different methods and programs were created for this purpose. One of these programs is discussed in this thesis. For this security analysis, Cryptomator, a tool for disk encryption, has been chosen.

The thesis consists of two parts; theoretical and practical.

The first part of the thesis aims to introduce the reader to the problem of storage security and commonly used technologies to achieve such a goal. The encryption techniques described there relate solely to the data that is stored (on disk or in memory), not to the transmitted data.

The opening section describes files and file systems; the next one provides

an overview of the encryption methods themselves – encrypting the entire disk, virtual disks, volumes, and encrypting individual files or folders. Finally, all the above methods are compared. The last section provides readers with a brief overview of the current options of storage encryption software, their features, and capabilities.

The practical part consists of Chapter 2 and Chapter 3.

Chapter 2 describes in detail the selected application, its advantages, disadvantages, and the differences from other disk encryption software. It discusses the architecture and technologies used in Cryptomator.

Chapter 3 is the analysis itself. The first two sections of this chapter analyze the UI and the source code of the application from a security point of view. The third section focuses on verifying the documented specifications by reimplementing Cryptomator's critical functions by using independent cryptographic libraries.

Chapter **1**

Storage Security

In today's digital life, it is essential to think about the security of data that is potentially put in risk not necessarily by intentional threats; the user himself often causes the damage unintentionally. [5]

So the question is, how can our data be protected? Achieving data safety is possible by using various encryption and authentication (identity verification) techniques. Data can be encrypted individually by files, but also in bulk (all stored data). The effectiveness of a given method is affected by many factors. You can also choose encryption for a single user (where others do not have access to encrypted data) or multiple users. User authorization may require a password, a PIN, or a biometric data. Regular file backups are also recommended. [5]
Before the introduction of each encryption method, it is necessary to define basic terms such as file or file system.

1.1 File and File System

This section aims to introduce the reader to fundamental terms, like file or file system. Both definitions are essential when talking about storage systems and storage encryption. Section 1.1.1 defines a file, a basic unit for data storage. Section 1.1.2 presents a systematic way for grouping files.

1.1.1 File

A file can be defined as a set of grouped information, with each file having its unique name. From the end-user perspective, there are two basic types of files – data files, executables, and system files needed for OS functionality. A data file can be a text, an image, audio, but also video, compressed data. The user accesses the data files directly, while the operating system mainly handles the system files. The specific file type is determined by its extension or a magic number that is stored inside the file. [5, 6]

1.1.2 File System

A file system defines the way files are stored and managed. Files are stored in a file system in folders that are used to organize them. Many operations can be done on a file system (such as folder creation, moving files between folders, or changing access rights). Different file systems use different folder structures. Modern file systems use a hierarchical directory system, depicted in Figure 1.1. [5, 6]

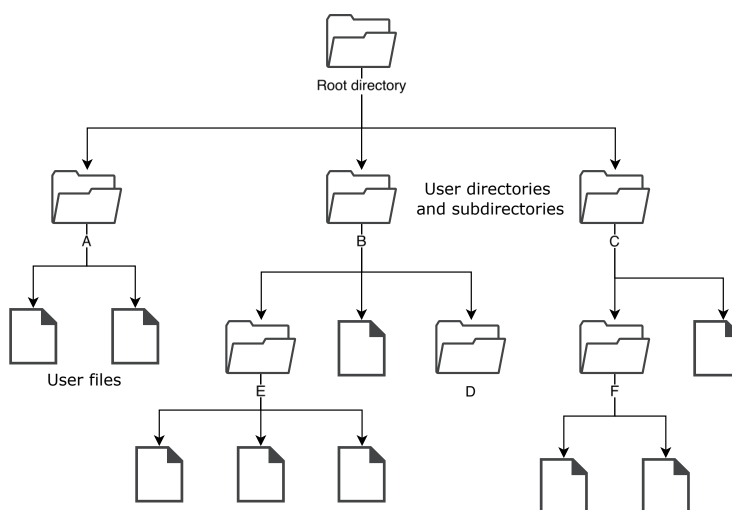


Figure 1.1: Example of a hierarchical directory system

The file system includes [6, 7]:

- **Space management** – organization of files and folders and their assigned physical space on the storage device; fragmentation can occur when data on the storage is fragmented; it is better to ensure that their positions are contiguous

- **Folders** – folders help keep track of files in the files system, allowing the user to store data in a hierarchical order. This approach enables grouping files in a systematic way, for example, when there are multiple users in the system, and each user can have its own root directory

- **File/Folder names** – naming helps distinct FS objects from each other. There may be restrictions on using certain characters or words in the name

- **Metadata** – contains information about files and folders; such as type, name, size, creation date, modification date, ownership, and access rights

- **Access control** – access rights and permissions; these can be assigned to a specific user or group of users. For most users, this method of control is sufficient, but it is not resistant to advanced attacks (for example, privilege escalation, when a SUID executable or a kernel is exploited).

Storages also contain information about already deleted or temporary files (residual data). The deleted file is not physically deleted from the storage, only the place where it is located is marked as free by the file system and can be overwritten. Therefore, even a free space can contain sensitive data. [5]

1.2 Encryption technologies

Stored data can be encrypted using different technologies. The most commonly used encryption methods are described in the following sections.

1.2.1 Full Disk Encryption

Full Disk Encryption (FDE) encrypts all data on the hard disk, including the operating system and all applications with user data, so that the files can only be accessed after authentication through the FDE component. The technology can be implemented either in software or hardware.

A partition table is a sector on media that serves as a part of the first sector on a storage device. It stores information about the storage partition scheme and decides which operating system to run (usually referring to the primary OS). The FDE software redirects the partition table to a special pre-boot environment (PBE) to verify the user's identity.

Verification can be done using an access name and password, an encryption key stored on a USB, tokens generating a one-time password, a fingerprint reader, and the like. Subsequently, FDE decrypts the OS boot sector, which starts to boot. The FDE software gradually decrypts the files in memory on-the-fly as needed by the user. This also applies to stored data that is decrypted.

Figure 1.2 shows a boot sequence on a storage with FDE.

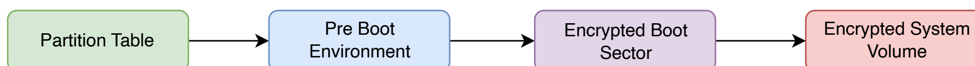


Figure 1.2: Boot Sequence with FDE technology [5]

However, the process of continually encrypting and decrypting data can slow down computer's work, which should only be noticeable with larger files. A delay of seconds also occurs when the computer is turned on or off, or when the computer enters/exits sleep mode. Other disadvantages of FDE may be

due to a change in the way the operating system boots, where modifying the partition table can cause problems with the correct startup of the computer, and there is also a risk of conflict with disk management software tools. [5]

In addition to the software solution, FDE can also be solved in hardware, where FDE is part of the hard disk controller. Also, in this case, authentication is required before loading the operating system, but the encryption code is stored directly in the firmware which is stored in ROM. There is no way to remove the FDE from it without destroying the device.

Although the whole disk encryption method can be considered secure, there are also some security risks in the form of password leakage, encryption key stored on USB (accessible), theft of a laptop with an already decrypted OS, and last but not least it also stores the encryption key in the memory of the running system (for example, in DRAM, where it is still available for some time after shutdown). By cooling the DRAM, this time can be extended, and the encryption key obtained. This type of attack is called a cold boot attack. [5, 8]

1.2.2 Virtual Disk Encryption

The method of virtual disk encryption (VDE) creates a special file on the disk, which at first glance looks like one unit – the so-called container that holds folders and files. A virtual disk represents the container. Access to the virtual disk requires software that runs under the operating system and evaluates all requests to read from or write to the container. Only after identity verification (and it can be in similar ways as in the previous case – for example, by entering a password) is the virtual disk, including the data on it, accessible. The required files are decrypted and encrypted on-the-fly based on the current requirements. There is also a variant where the virtual disk is automatically made available after successful user authorization.

The virtual disk can be copied without the need to decrypt it, so it is portable and usable on other media, which makes it easier to backup. Here again, an authentication process is required to decrypt the content. Furthermore, it is appropriate to consider the configuration of the operating system, i.e., whether it is allowed for the user to write files only to the encrypted virtual disk or not. [5]

1.2.3 Volume Encryption

Volume Encryption works on a similar principle to VDE, but instead of a virtual disk, only a logical volume, which is typically located on an internal storage device or removable storage media (such as a USB flash drive), is encrypted. Access to the information on the volume is possible again after the authentication process. Again, the software is needed that controls access to

the volume and, based on successful user identification, decrypts and encrypts the data on the volume that the user wants to read or write.

In addition to regular data, system or boot volumes can also be encrypted. Here, like in FDE, the possibility of SSO can be used, where encrypted volumes are automatically made available to an authorized user. [5]

1.2.4 File and Folder Encryption

File Encryption ensures that individual files in memory or on disk are protected, as in previous cases, by the authentication process. A user can encrypt files as well as entire folders (Folder Encryption) – and while it can be said that the principle is the same, folder encryption must also manage the encryption of metadata related to the folder contents. Encryption is possible at the operating system level, but a number of third-party programs are also available. [5]

At first look, it appears that the method of encrypting folders is similar to the method of encrypting virtual disks, which can also contain a number of files (as well as a folder), but there is a difference. The contents of the virtual disk are not visible without the necessary user authentication, so those who have not been authorized cannot see what specific folders or files are on it. On the other hand, the files in the folder that is encrypted are visible (their names or even metadata can be displayed), only it is not possible to access their contents. This encryption method applies to any type of storage media. [5]

The decryption process is the same as in the previous cases – after successful user authentication, the file gets decrypted and opened. Because individual files get decrypted, this form of security has minimal effect on their speed of opening. There are several options when encrypting files and folders. Firstly, it is up to the user to decide which files or folders he wants to encrypt; the administrator can specify automatic encryption of the contents of specific folders. It is also possible to encrypt only specific file types (based on their extension or magic number), you can automatically encrypt all files that are created by specific software, and finally, any files that the user creates. [5]

This method has one indisputable advantage: if the entire disk is encrypted and data and software on it are made available, the individual files may be not protected. This can be solved by encrypting files or folders. However, there is also the option to set up automatic decryption of all specified files after the authentication of a user.

When encrypting files, keep in mind that the file may not be completely protected, as the residual data described in Section 1.1.2 may also be associated with it, and it is not protected. An additional issue with this method can be that before using a file, it must be decrypted, which is not on-the-fly operation and may take some time. It can also lead to residual data creation. [5, 8]

1.3 Comparison of encryption methods

When picking data encryption technologies, it must be considered how each technique will change the end-user devices and infrastructure. A comparison of these data encryption methods is provided in Table 1.1, which is based on the information provided by the NIST Special Publication 800-111. [5]

Characteristic	**Encryption technology**			
	Full Disk Encryption	Virtual Disk Encryption	Volume Encryption	File/Folder Encryption
Supported Devices	Desktop machines	All types of devices	Desktop machines, removable storage devices	All types of devices
Scope of protected data	All data on storage (user/system files, metadata, and residual data)	All data in container (user data, metadata, and residual data)	All data in container (user data, metadata, and residual data)	Specific files/folders
Potential data loss in case of a storage failure	All data on device is lost	All data in container is lost	All data in volume is lost, may damage storage functionality	All encrypted files/folders are lost
Encrypted data portability	Portable	Portable	Portable	Portable

Table 1.1: Comparison of encryption methods

1.4 Existing tools for encryption

Nowadays, a relatively large amount of software that offers secured access to data can be found on the market. In addition to data encryption, the other most implemented functionalities, among others, are generating random passwords, password recovery options, hardware token support, and a choice of various encryption algorithms.

The following sections briefly describe some of the alternatives for data encryption available for users on computer machines.

1.4.1 BestCrypt

BestCrypt is a commercial disk encryption software developed by Jetico. BestCrypt offers two variants: Container Encryption and Volume Encryp-

tion. BestCrypt works with a wide range of encryption algorithms, including DES/Triple DES, AES, Serpent, Blowfish, Twofish with support of CBC, XTS, and LRW modes of operation. It is worth noting the support of hardware tokens, such as YubiKey and SafeNet eToken. [9]

1.4.2 FileVault 2

FileVault 2 is a built-in macOS software made by Apple for whole-disk encryption that protects all data on a Mac and prohibits illegal access without the decryption key or the valid credential. FileVault 2 uses XTS-AES-128 and an encryption key with a size of 256 bits in compliance with NIST Special Publication 800-38E. FileVault 2 provides several password recovery options as well as an easy-to-use user interface. [10, 11]

1.4.3 LUKS

Linux Unified Key Setup (LUKS) is a disk encryption format standard, aimed initially at use in OSes based on the Linux kernel. The primary goal was to provide a user-friendly, standardized way of managing decryption keys.

One of the features of the standard is the support of several encryption keys used on an equal basis with each other to access a single encrypted medium, with the possibility of adding and removing them at the user's demand.

The LUKS1 and LUKS2 specifications define platform-independent standards that are available in the cryptsetup tool. It uses the dm-crypt tool as a backend for a disk encryption. [12]

1.4.4 VeraCrypt

VeraCrypt is a free, open-source storage encryption software. VeraCrypt originated as a fork of the discontinued TrueCrypt project. [13]

VeraCrypt provides the ability to use FDE (only on Windows), VDE as well as volume encryption. It also has the option for an encrypted outer volume to have an additional volume hidden inside. It's possible to have a hidden OS in such volume.

VeraCrypt supports multiple encryption algorithms such as AES, Camellia, Serpent. Cascade variants of these algorithms are available. Cascade encryption is a method of encryption when multiple (mostly two or three) different algorithms are used with different encryption keys. [14]

VeraCrypt mainly uses the XTS mode for encryption. Hardware tokens can be used as a keyfile provider. [15]

Chapter 2

Cryptomator

Cryptomator is an open-source disk encryption software specifically designed to work with cloud storage. Cryptomator is available for free on most desktop platforms. On mobile platforms, however, it costs a one-time fee with no extra microtransactions. [16, 17]

2.1 Features of Cryptomator

- **Multiplatform** – Cryptomator supports popular modern platforms: macOS, Windows, Linux, iOS, Android
- **Open-source** – Cryptomator for desktop is written in Java, and its source code is available for download and further study on GitHub [18]
- **Cloud storages support** – Cryptomator works with different cloud storages, such as Dropbox, iCloud Drive, Google Drive, One Drive
- **Modern encryption** – Cryptomator uses the latest standards for all encryption operations, including file/folder name encryption [19]
- **Multiple vaults support** – Cryptomator allows the user to use as many vaults as the user needs
- **File contents integrity checks** – Cryptomator architecture allows checking all encrypted content for integrity

2.2 Architecture

This section introduces the principles and technologies Cryptomator is designed upon. All encryption algorithms, their respected operation modes, and hash functions described in the following sections are discussed in Section 2.3, Cryptography.

2.2.1 Virtual File System

The main unit for storing encrypted user data is called a Vault. Vault is stored physically as a special folder consisting of a special file called masterkey.cryptomator and two inner folders d and m, d contains the actual encrypted user data. In contrast, m contains metadata related to user data from the d folder. Internally the Vault is implemented as a virtual drive by using one of the three frontends available to the user: FUSE (macOS or Linux), Dokany (Windows), and WebDAV as a fallback if the other two are not available. All the user-initiated operations (such as reading or writing a file) are made using this virtual drive. [19]

2.2.2 Masterkey File

Masterkey is a file unique to each Vault. It contains all information required for Cryptomator to work correctly with the Vault. At the moment of writing this thesis, the current version of the Masterkey file is version 6.

Masterkey file is a JSON object, containing [19]:

- **scryptSalt** – a Base64 string representing a salt data used for the scrypt key derivation function (KDF)
- **version** – a number representing a Masterkey version, in this case, it equals 6
- **scryptCostParam** – a constant used in the scrypt KDF. Equals to 16384. Used for CPU and memory optimizations.
- **scryptBlockSize** – a constant used in the scrypt KDF. Equals to 8
- **primaryMasterKey** – a Base64 string representing an encryption master key encrypted using the AES Key Wrap algorithm (RFC 3394)
- **hmacMasterKey** – a Base64 string representing a MAC master key encrypted using the AES Key Wrap algorithm (RFC 3394)
- **versionMac** – a Base64 string representing a HMAC-256 of the version value to stop Masterkey version downgrade attacks

2.2.3 Keys Initialization

When the user creates a Vault, and the structure described in Section 2.2.1 gets generated, a master key is initialized. First, a pair of encryption and MAC master keys is generated by a cryptographically secure pseudorandom number generator (CSPRNG). In this case, SHA1PRNG is used. The use of CSPRNG ensures the strength of the underlying cryptographic primitives, due to the unpredictability of generated bits. [20]

After that, both keys get enrypted using the AES Key Wrap algorithm. A key encryption key (KEK) is used for this operation. KEK is derived from a password entered by the user with the scrypt KDF. [19]

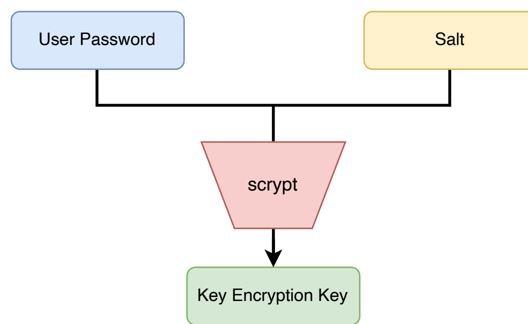


Figure 2.1: KEK derivation using the scrypt KDF

Both keys are then written into a Masterkey file, described in the previous section, for later use in operations with the Vault. The whole process of keys initialization is depicted in Figure 2.2.

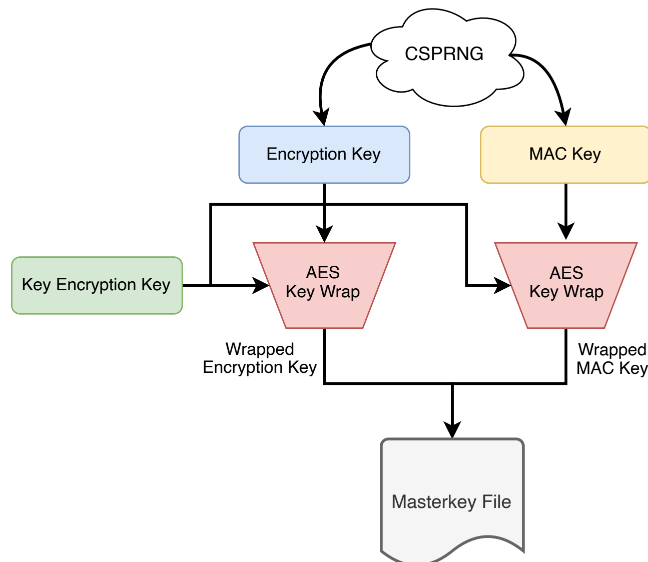


Figure 2.2: Process of key initialization

2.2.4 File/Folder Name Encryption

One of the key Cryptomator features is a file name encryption and folder structure obfuscation. In order to achieve that, both file and folder names are encrypted using AES with SIV mode of operation is used. For each folder, a unique ID is assigned. The root directory is an exception, with the unique ID equal to a zero bytes string. A file name is passed to the algorithm as is. AES-SIV uses the encryption and MAC key for encryption and IV synthesis, respectively. The parent folder's unique ID is also passed as additional data to the IV synthesis function; this mitigates unauthorized changes to the VFS structure. If it's a directory, a zero is then added at the begging of the encrypted name. [19]

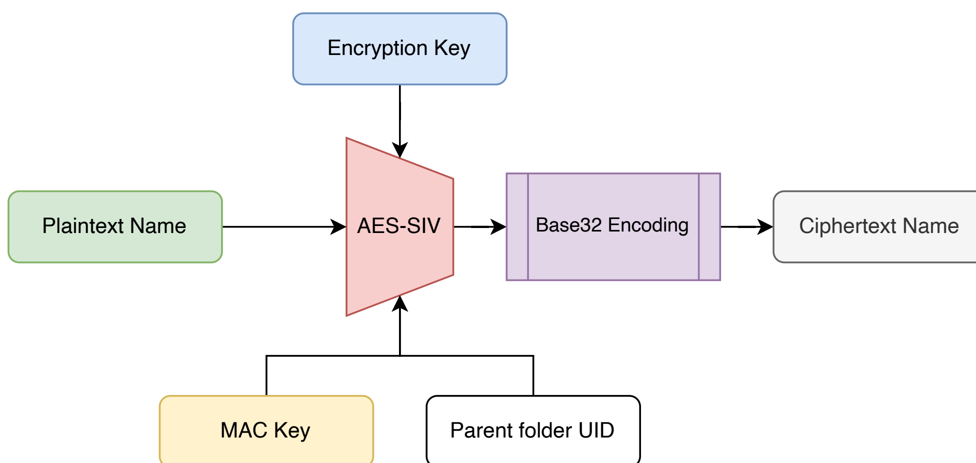


Figure 2.3: Process of file/folder name encryption [19]

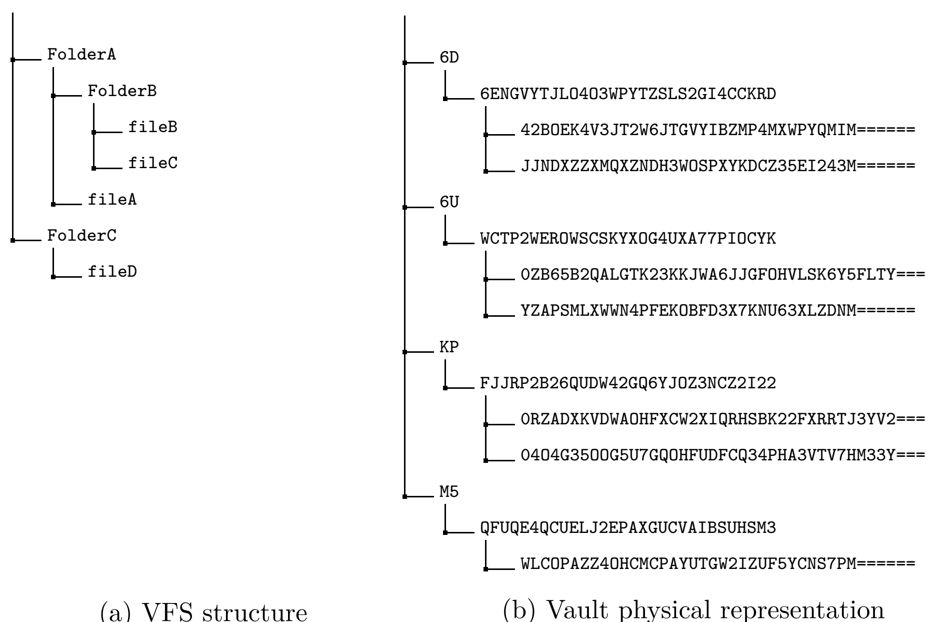
For example, with the encryption key set to:

5313A6E4 A100478E FF85F7A2 35BC374D
4FB5954D 9235B99D 529132F3 2BCF6F3C

and the MAC key equal to:

F1930AD7 528568E1 DA5417F4 626A1224
F9816261 3D3B7A67 C7315B57 CAE55E64 ,

the following folder structure would look like this:



(a) VFS structure (b) Vault physical representation

Figure 2.4: An example of FS structure transformation

2.2.5 File Header Encryption

This section describes the construction of a file header that contains metadata needed for decryption of the file content.

First, a random nonce is generated. It's unique for each file and later used for the File Content Key (FCK) encryption and the integrity checks of data chunks. The File Content Key consists of 32 bytes of random data and is encrypted with 8 bytes set to 1's (0xFF×8) using AES-CTR. After that, a HMAC-SHA256 of the nonce and encrypted FCK is calculated and placed at the end of the header. [19]

The table below shows the layout of a file header.

offset	size	description
0	16	Random nonce used as IV for the header encryption and the file integrity checks
16	40	AES-CTR of: 8 bytes filled with 1 (0xFF×8) + 256 bit File Content Key
56	32	HMAC-SHA256 of the previous 56 bytes

Table 2.1: Layout of the file header

The total size of the header is 88 bytes.

2.2.6 File Content Encryption

Before the actual encryption happens, the file data gets split down into chunks of up to 32KiB. Each chunk also has an additional data of 48 bytes required for the encryption process and the integrity checks. The whole process of a file data encryption is depicted in Figure 2.5. [19]

The table below shows the layout of a file chunk.

offset	size	description
0	16	Random nonce used as IV for the chunk encryption and the integrity checks
16	N	Up to 32KiB of the chunk data, encrypted with AES-CTR
16+N	32	HMAC-SHA256 of the file header nonce, the chunk index, the chunk nonce, and the encrypted data

Table 2.2: Layout of the file chunk

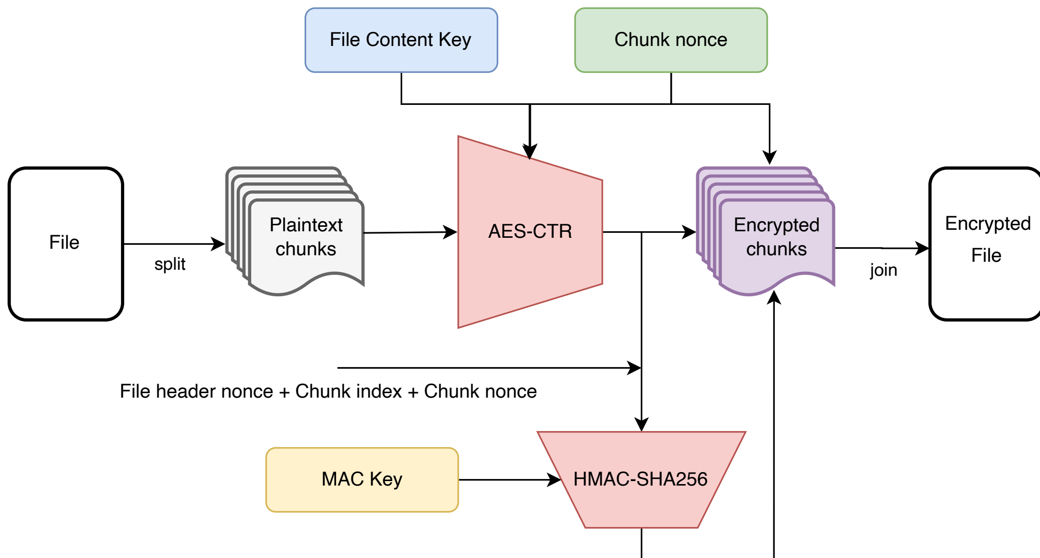


Figure 2.5: Process of file content encryption [19]

2.3 Cryptography

This section defines the AES cipher, various modes of operation, and cryptographic hash functions used in Cryptomator. Algorithm descriptions are provided for each cryptographic primitive.

2.3.1 AES

Currently, only the AES cipher is supported. Developed by Vincent Rijmen and Joan Daemen, AES is a symmetric block cipher with a block length of 128 bits and a key size of 128, 192, or 256 bits. [21]

Cryptomator supports only AES-256, which means a key size must equal 256 bits.

AES-256 is described in the pseudo-code below.

Algorithm 1: AES-256 [21]

```
Input: key K, data D
Result: Encrypted D using AES-256
begin
    noOfRounds ← 14;
    state ← D;
    expKey ← KeyExpansion(K);
    state ← AddRoundKey(state, expKey, 0);
    /* first 13 rounds                                  */
    for i = 1; i < noOfRounds; i++ do
        state ← SubBytes(state);
        state ← ShiftRows(state);
        state ← MixColumns(state);
        state ← AddRoundKey(state, expKey, round);
    end
    /* last round                                       */
    state ← SubBytes(state);
    state ← ShiftRows(state);
    state ← AddRoundKey(state, expKey, round);
    return state
end
```

- **KeyExpansion** is a function for generating round keys from the encryption key [21]
- **AddRoundKey** is a function that adds a round key to the state by using XOR operation [21]
- **SubBytes** is a function for a non-linear byte substitution operating separately per byte using the S-box table [21]
- **ShiftRows** is a function that processes the state, cyclically shifting the last three rows of the state by different values [21]
- **MixColumns** is a function that takes all the state columns and mixes their values (individually) to get new columns [21]

2.3.2 Modes Of Operation

Cryptomator uses only two modes of operation: CTR and SIV. CTR mode is used for file's content encryption, while SIV is needed for file/folder name encryption. The following sections describe both of these modes.

2.3.2.1 CTR

CTR mode turns AES into a stream cipher. It generates the keystream block by encrypting the sequential values of the so-called counter. A counter can be any function that produces a sequence that is guaranteed not to repeat for a long time. The encrypted value is then XOR'ed with a plaintext block to get ciphertext. [22]

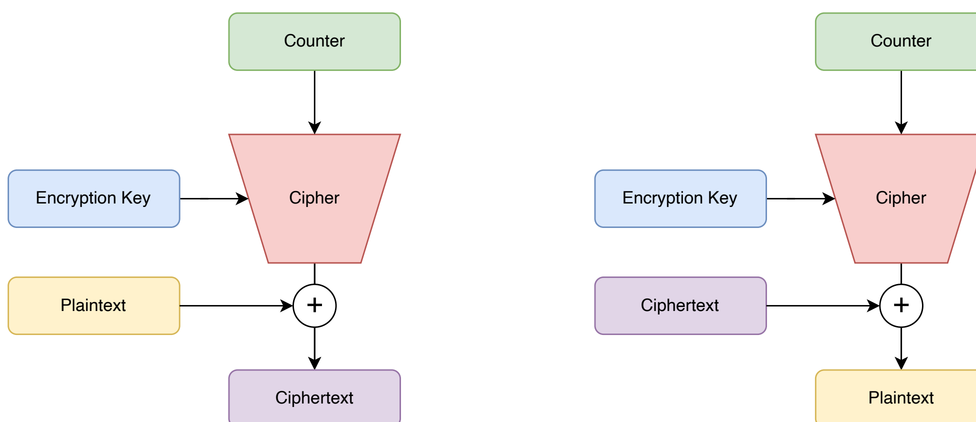


Figure 2.6: Encryption and decryption with CTR mode

2.3.2.2 SIV

SIV is an authenticated mode of operation for AES. It combines the CTR mode with AES-CMAC for integrity protection.

SIV takes a key, a plaintext, and optional additional data that is used for authentication. The result is a synthetic initialization vector (SIV) and a ciphertext with the same length as the plaintext. [23]

Listings below describe the operation of the mode.

Algorithm 2: AES-SIV [23]

Input: Encryption key K_E, MAC key K_M, plaintext P, data $S_1, ..., S_n$
Result: Ciphertext with SIV
begin
  IV ← $S2V(K_M, S_1, ..., S_n, P)$;
  `/* || is concatenation */`
  C ← IV & (1^{64} || 0^1 || 1^{31} || 0^1 || 1^{31}); `/* counter */`
  X ← AES-CTR(K_E, IV, C, P);
  return IV || X;
end

Algorithm 3: S2V [23]

Input: MAC key K_M, data $S_1, ..., S_n$
Result: Synthetic IV
begin
  if $n = 0$ **then**
    `/* || is concatenation */`
    return AES-CMAC(K_M, 0^{127} || 1^1);
  end
  D ← AES-CMAC(K_M, 0^{128});
  for $i = 1; i < n; i{+}{+}$ **do**
    D ← dbl(D) ⊕ AES-CMAC(K_M, S_i);
  end
  if $length(S_n) \geq 128$ **then**
    T ← S_n ⊕ D;
  else
    T ← dbl(D) ⊕ pad(S_n);
  end
  return AES-CMAC(K_M, T)
end

2.3.3 scrypt

In cryptography, password-based key derivation functions are used for deriving secret keys from a secret value, in this case, a password.

scrypt is a key derivation function developed by Colin Percival. Designed to be resistant to brute force and custom hardware attacks, it uses memory-hard functions to achieve that goal. [24]

Under the hood, scrypt uses the well-known PBKDF2 (with HMAC-SHA256) created by RSA Laboratories and Salsa20/8.

Algorithm 4: scrypt [24]

```
Input: passphrase P, salt S, cost factor N, block size factor r,
       parallelization factor p, key length l
Result: key derived from P
begin
    blockSize ← 128 * r ;                              /* in bytes */
    B ← PBKDF2(P, S, 1, blockSize * p);
    for i = 0; i < p; i++ do
        B_i ← ROMix(B_i, N);
    end
    /* || is concatenation                                       */
    newSalt ← B_0 || B_1 || ... || B_{p−1};
    return PBKDF2(P, newSalt, 1, l)
end
```

ROMix is a sequential memory-hard function that makes the creation of custom hardware an expensive task. The function computes a large number of pseudo-random values and then stores them in RAM. [24]

More detailed explanation of the scrypt KDF can be found in the white paper written by the author of the algorithm. [24]

2.3.4 HMAC

HMAC is a message authentication mechanism using cryptographic hash functions. Any cryptographic hash function with iterative nature (such as SHA-1, SHA-256, SHA-512) can be used in HMAC. The strength of HMAC directly depends on the strength of the used hash function. [25].

Algorithm 5: HMAC-SHA256 [25]

```
Input: key K, data D
Result: HMAC-SHA256 of D
begin
    blockSize ← 64;                                   /* in bytes */
    hashSize ← 256;                                    /* in bits */
    ipad ← 0x36 * blockSize;  /* inner padding intialization */
    opad ← 0x5C * blockSize; /* outer padding intialization */
    if length(K) > blockSize then
        K ← SHA-256(K);
    end
    for i = 0; i < length(K); i++ do
        ipad[i] ← ipad[i] ⊕K[i];
        opad[i] ← opad[i] ⊕K[i];
    end
    /* || is concatenation                                     */
    return SHA-256(opad || SHA-256(ipad || D))
end
```

In the case of Cryptomator, SHA-256 is used for calculating HMAC.

SHA-256 is built on the Merkle–Damgård structure. After the padding, the plaintext is split into blocks, each block into 16 words. The algorithm passes each block of the message through a cycle with 64 rounds. At each iteration, 2 words are transformed; the remaining words define the transformation function. The processing results of each block are added; the resulting sum is the value of the hash function. [26]

Chapter 3

Security Analysis

The goal of this chapter is to provide a summary of the security analysis of Cryptomator. The analysis was conducted on macOS using Cryptomator version 1.4.17. The report consists of three parts: the UI analysis, the source code analysis, and the reimplementation based on the provided documentation. In the process, multiple weaknesses were discovered in different parts of the program. Each found problem is described in detail with possible solutions advised.

3.1 UI Analysis

This section talks about the UI of Cryptomator, process of the Vault creation, password managing, and related potential security problems discovered while studying the UI.

3.1.1 Main Screen

When Cryptomator is launched, a user is presented with a welcome screen. By pressing a cog button, settings pane (Figure 3.1) shows up. The settings pane allows users to chose preferred frontend for the VFS (as well as some configurations for the selected frontend, e.g., WebDAV port if WebDAV is selected), enable automatic check for updates, and debug mode. By enabling debug mode, Cryptomator creates additional log files with useful information for solving problems with a vault.

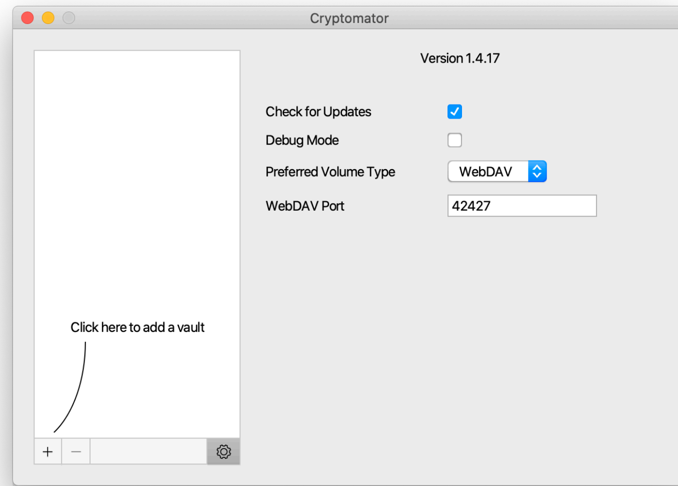


Figure 3.1: Settings pane

3.1.2 Vault Management

By pressing the "+" button, two options are presented: create a new Vault, or add already existing Vault. A pane where the user can enter the Vault password is presented when creating a new Vault.

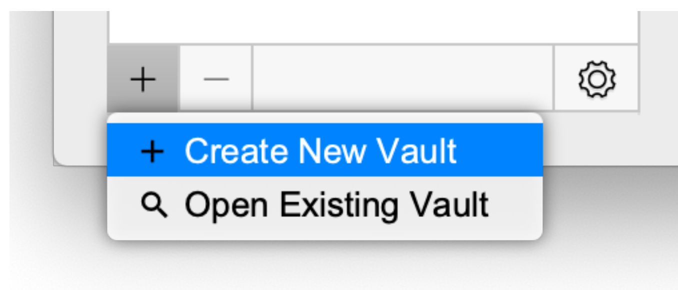


Figure 3.2: Vault management options

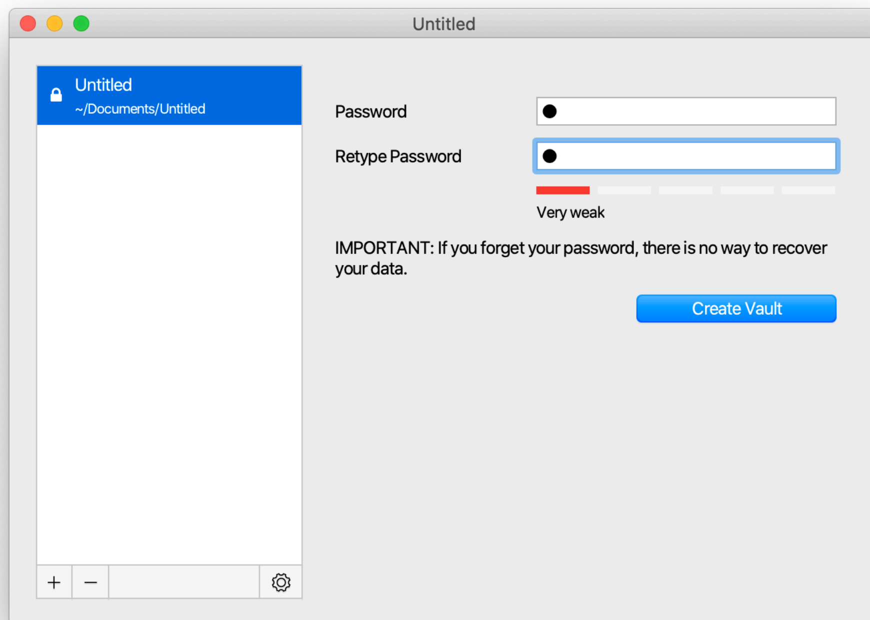


Figure 3.3: Password input when creating a Vault

When studying this pane, several problems and disparities with the NIST standard [27] were discovered:

- In accordance with the standard's recommendations, the minimum password length selected by the user must be at least eight characters
- When confirming the password, there is no additional confirmation layer when using a weak password
- It is also recommended to provide the user with a function to disable the hiding of the password with dots to be able to verify the correctness of the entered password
- There is no way to create a recovery key when the password is forgotten, or there is no way to enter it

While these flaws are not critical, they indirectly reduce the safety of the user data. For more effective security of the data, a strong password is required.

One of the indicators of password strength is the entropy of this password. Entropy is expressed in bits and defines the value of password ambiguity. [28]

Entropy H of a password of length l using N number of character can be calculated using this formula [28]:

$$H = \log_2(N^l)$$

Usage of this formula can confirm that a randomly chose password of eight characters (which is recommended by the standard) generated using a set of 94 printable ASCII characters (excluding space) provides entropy equal to 52.7 bits. It is twice as better entropy as the password in the same characters set with a length of four characters, and 1.4 times better entropy than a password consisting of six characters. [28]

3.1.3 Miscellaneous UI related problems

Problems with changing the password were found, in addition to those described earlier in the previous section.

- Menu for changing a password is hidden and hard to find, making it difficult for the user to update the password in case of a compromise of the old one

- The Change password pane doesn't notify the user that files won't get re-encrypted, and the Vault will be vulnerable to the masterkey downgrade attack. That allows an attacker to reuse the older version of masterkey with a weak password to unlock the Vault. The attack itself is described in detail in Section 3.2.1

Also, a problem with the "Save password" option exists. The saved password is not removed from a keychain when the option is turned off, or the Vault is removed – allowing an attacker to recover a saved password from a keychain.

A solution could be removing a password from a keychain after a Vault removal or after turning the option off.

3.2 Source Code Analysis

During the study of the application's architecture, it was concluded to focus during the source code analysis on the implementation of VFS, UI, and the cryptographic parts of Cryptomator. [29, 30]

After examining crucial parts of the code, three potential security problems were found. These problems are described in the following sections. Nevertheless, the studied cryptographic primitives are implemented correctly and in accordance with the relevant standards.

While studying the code, it was noted that the Zxcvbn algorithm by Dropbox is used to check the password complexity. A detailed study of this algorithm and its comparison to other similar algorithms was made by Xavier de Carné de Carnavalet and Mohammad Mannan. [31]

Additionally, independent audits of the code were conducted by Cure53 and Tim McLean. [32, 33]

3.2.1 Masterkey Replay Attack

One of the problems found during the analysis was that changing a Vault password would only change the respective KEK and would not change the Vault's encryption key, making it an easy target for a master key replay attack.

Given that Cryptomator is intended for use with cloud storage, an attacker, who has access to the user's cloud storage, can quickly recover an older version of the master key with a weak password and use it for unlocking the newly updated Vault. The example of this potential attack is depicted below.

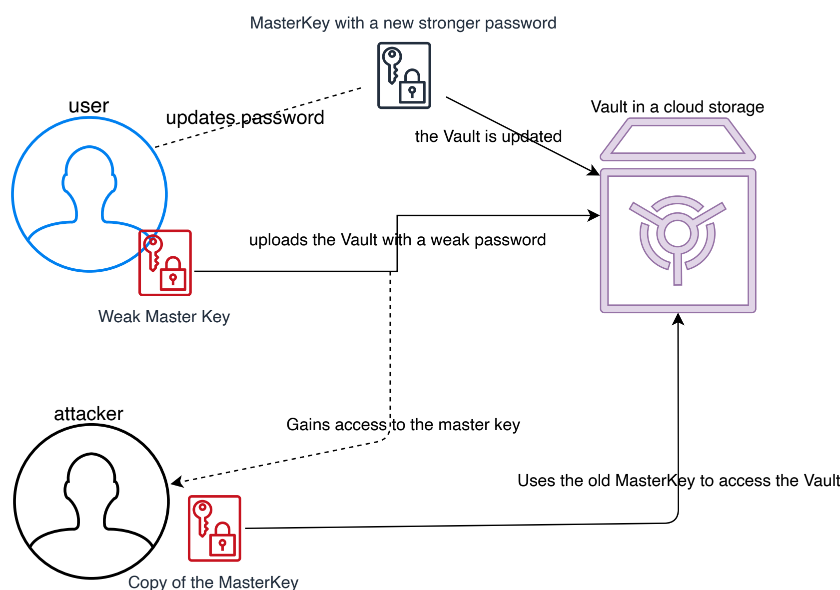


Figure 3.4: Example of the masterkey replay attack

Currently, the only effective way to change a password without threatening data would be to create a new Vault from scratch. One way to solve this problem is to re-generate the encryption keys and sequentially re-encrypt the whole storage with a new key.

3.2.2 File Contents Swap

In this illustration, let us assume there are two different files in the same folder. Because the filename isn't linked to the file content (see Section 2.2.4),

those two files' contents can be swapped.

Hence an attacker can change the files' contents, hoping for the user to publish decrypted versions of secret files accidentally.

A solution to this problem would be enforcing integrity checks for the file's name and its content.

3.2.3 Password persistence in RAM

It was also noted that the Vault's password is stored in RAM even after closing this Vault. In some cases, password information remains in memory even after the program is closed. This weakness can be exploited by using a cold boot or DMA attack. [34, 35]

A possible solution to the problem is to provide an option to encrypt RAM sections as VeraCrypt does. [15]

At the same time, RAM encryption is not a complete solution to the problem, since RAM encryption raises another question: how to protect the encryption key stored in an unencrypted memory area? In order to secure the key, it can be stored safely in a TPM chip or similar hardware (such as T2 Security Chip).

3.3 Implementation

This section focuses on the reimplementation of the core functionalities of Cryptomator. The purpose of this implementation was to check if the source code and executables provided by the Cryptomator team honor the program's documentation and if there are any deviations in the program functioning.

Swift and Objective-C languages were chosen to verify the implementation. Specifically, Objective-C was used for SIV mode of operation implementation and essential system operation related to encryption. All other parts of the program are written in Swift.

At the moment of writing this thesis, the code is compilable only on a machine running macOS due to the exclusivity of Objective-C on macOS/iOS systems. After making some alterations to the code written in Objective-C (such as AES-SIV and Base32 related code), it would also be possible to compile the reimplementation on Linux.

Standard libraries provided by macOS were chosen for this reimplementation due to the open-source nature of Swift and Apple system libraries. [36, 37] The following libraries and 3rd party code were also used:

- **libscrypt-kdf** by Colin Percival is used for a password-based KDF scrypt [38]

- **Base32Additions** by Dave Poirier is an RFC 4648 compatible implementation of Base32 encoding [39]

3.3.1 Verification of correspondence

During the verification process, all critical operations, such as Vault creation, Vault unlocking with the following parsing, file/folder encryption, and decryption, were successfully implemented. All vaults created and managed by Cryptomator were accessible with the reimplementation and vice versa.

Based on the results of the implementation testing, it's clear that Cryptomator conforms to its documentation. A total number of 50 different tests were conducted.

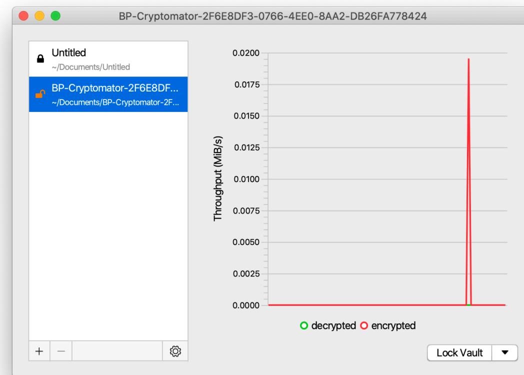


Figure 3.5: Cryptomator working with a vault created by the reimplementation

3.4 Update 1.5.0

Just before the security analysis was completed, the Cryptomator team released a new update, version 1.5.0. After careful examination it was noticed that the version 1.5.0 fixes some of the issues found during the conducted security analysis.

These fixes include:

- Minimum length of a password is set to eight
- Added the ability to show password when typing
- It's now possible to create a recovery key in the case when the password is lost
- It's now easier to change a password

Conclusion

This thesis aimed to introduce the reader to the disk encryption issue and the Cryptomator tool, including its security analysis. During the study of Cryptomator, I learned how encryption software works and expanded my knowledge of cryptography.

The first chapter described storage encryption theory. The best-known methods and their comparison have also been described in this chapter. The most popular software was outlined at the end of the chapter.

The purpose of the second chapter was to describe Cryptomator, its capabilities, and its architecture. The cryptographic primitives used in Cryptomator were defined later in the chapter.

The last chapter dealt with the analysis of program security from different angles: the UI, analysis of the essential parts of the source code, and verification of compliance between the Cryptomator's documentation and its implementation.

During the analysis, several problems were found in different parts of the program and described in this thesis. One of these problems is weak password strength checks and recommendations that oppose ones provided by related standards. Insecure handling of password change for a Vault and file integrity checks were discovered. Own re-implementation using the provided documentation did not reveal any inconsistencies with the Cryptomator's implementation and the provided documentation.

The more complex a system becomes, the more difficult it is to make it safer, so it is vital to conduct security analyzes on a continuing basis. [40] In the future, it might be possible to expand the thesis with a more comprehensive analysis of future updates and a closer examination of the source code.

Bibliography

[1] Fournier, E. Why Cloud Storage is Growing in Use and Popularity. Available from: https://articles.bplans.com/why-cloud-storage-is-growing-in-use-and-popularity/

[2] Universal declaration of human rights. *UN General Assembly*, volume 302, no. 2, 1948. Available from: https://www.un.org/en/universal-declaration-human-rights/

[3] Ruiz, D. There is No Middle Ground on Encryption. *Electronic Frontier Foundation (EFF)*, 2018. Available from: https://www.eff.org/deeplinks/2018/05/there-no-middle-ground-encryption

[4] Galperin, E.; Kayyali, N.; et al. Dear NSA, Privacy is a fundamental right, not reasonable suspicion. *Electronic Frontier Foundation (EFF)*, 2014. Available from: https://www.eff.org/deeplinks/2014/07/dear-nsa-privacy-fundamental-right-not-reasonable-suspicion

[5] Scarfone, K.; Souppaya, M.; et al. Guide to storage encryption technologies for end user devices. *NIST Special Publication*, volume 800, 2007: p. 111. Available from: https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-111.pdf

[6] Stallings, W. *Operating Systems: Internals and Design Principles.* Pearson Education, 2018.

[7] Tanenbaum, A. S.; Bos, H. *Modern operating systems.* Pearson, 2015.

[8] Rubens, P. Full Disk Encryption Buyer's Guide. *eSecurity Planet*, 2012. Available from: https://www.esecurityplanet.com/mobile-security/buyers-guide-to-full-disk-encryption.html

[9] Data Encryption | Jetico. 2020. Available from: https://www.jetico.com/data-encryption

[10] Use FileVault to encrypt the startup disk on your Mac. *Apple Support*, 2018. Available from: https://support.apple.com/HT204837

[11] Best Practices for Deploying FileVault 2. *Apple Technical White Paper*, 2012.

[12] cryptsetup / cryptsetup · GitLab. *GitLab*, 2020. Available from: https://gitlab.com/cryptsetup/cryptsetup/

[13] VeraCrypt - Free Open source disk encryption with strong security for the Paranoid. 2020. Available from: https://www.veracrypt.fr/en/Home.html

[14] Gaži, P.; Maurer, U. Cascade Encryption Revisited. Available from: https://www.iacr.org/archive/asiacrypt2009/59120035/59120035.pdf

[15] VeraCrypt - Free Open source disk encryption with strong security for the Paranoid. 2020. Available from: https://www.veracrypt.fr/en/Documentation.html

[16] Cryptomator - Free Cloud Encryption for Dropbox & Co. *Cryptomator*, 2020. Available from: http://cryptomator.org

[17] FAQ. *Cryptomator*, 2020. Available from: http://cryptomator.org/faq/

[18] Cryptomator · GitHub. *GitHub*, 2020. Available from: https://github.com/cryptomator/

[19] Architecture - Cryptomator. *Cryptomator*, 2020. Available from: https://docs.cryptomator.org/en/1.4/security/architecture/

[20] Aumasson, J.-P. *Serious cryptography: a practical introduction to modern encryption*. No Starch Press, 2017.

[21] Announcing the ADVANCED ENCRYPTION STANDARD (AES). *Federal Information Processing Standards Publication*, volume 197, 2001. Available from: https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197.pdf

[22] Dworkin, M. Recommendation for Block Cipher Modes of Operation: Methods and Techniques. *NIST Special Publication*, volume 800, 2001: p. 38A. Available from: https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-38a.pdf

[23] Harkins, D. Synthetic Initialization Vector (SIV) Authenticated Encryption Using the Advanced Encryption Standard (AES). *Request for Comments: 5297*, 2008. Available from: https://tools.ietf.org/html/rfc5297

[24] Percival, C. STRONGER KEY DERIVATION VIA SEQUENTIAL MEMORY-HARD FUNCTIONS. 2009. Available from: http://www.tarsnap.com/scrypt/scrypt.pdf

[25] Krawczyk, H.; Bellare, M.; et al. HMAC: Keyed-Hashing for Message Authentication. *Request for Comments: 2104*, 1997. Available from: https://tools.ietf.org/html/rfc2104

[26] Secure Hash Standard (SHS). *FIPS PUB 180-4*, 2015. Available from: https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.180-4.pdf

[27] Grassi, P.; Fenton, J.; et al. Digital Identity Guidelines: Authentication and Lifecycle Management. *NIST Special Publication*, volume 800, 2020: p. 63B. Available from: https://pages.nist.gov/800-63-3/sp800-63b.html

[28] Grassi, P.; Fenton, J.; et al. Digital Identity Guidelines. *NIST Special Publication*, volume 800, 2020: pp. 63–3. Available from: https://pages.nist.gov/800-63-3/sp800-63-3.html

[29] GitHub - cryptomator/cryptolib: Cryptomator Crypto Library. *GitHub*, 2020. Available from: https://github.com/cryptomator/cryptolib

[30] GitHub - cryptomator/cryptomator: Cryptomator. *GitHub*, 2020. Available from: https://github.com/cryptomator/cryptolib

[31] de Carné de Carnavalet, X.; Mannan, M. From very weak to very strong: Analyzing password-strength meters. In *Network and Distributed System Security Symposium (NDSS 2014)*, Internet Society, 2014. Available from: https://www.ndss-symposium.org/wp-content/uploads/2017/09/06_3_1.pdf

[32] Heiderich, M.; Kobeissi, N. Pentest-Report Tresor Application Crypto. *Cure53*.

[33] McLean, T. SIV Encryption Security Assessment. 2016. Available from: https://www.chosenplaintext.ca/publications/20161104-siv-mode-report.pdf

[34] Pilkey, A. The Chilling Reality of Cold Boot Attacks. 2018. Available from: https://blog.f-secure.com/cold-boot-attacks/

[35] Blocking the SBP-2 driver and Thunderbolt controllers to reduce 1394 DMA and Thunderbolt DMA threats to BitLocker. 2018. Available from: https://support.microsoft.com/en-gb/help/2516445/blocking-the-sbp-2-driver-and-thunderbolt-controllers-to-reduce-1394-d

[36] Apple · GitHub. *GitHub*, 2020. Available from: https://github.com/apple/

[37] Apple Open Source. *Apple*, 2020. Available from: https://opensource.apple.com

[38] Tarsnap - The scrypt key derivation function and encryption utility. 2020. Available from: http://www.tarsnap.com/scrypt.html

[39] Poirier, D. Base32 Additions for Objective-C on Mac OS X and iOS. 2020. Available from: https://github.com/ekscrypto/Base32

[40] Schneier, B. A Plea for Simplicity. *Schneier on Security*, 1999. Available from: https://www.schneier.com/essays/archives/1999/11/a_plea_for_simplicit.html

Appendix A

Acronyms

AES Advanced Encryption Standard

ASCII American Standard Code for Information Interchange

CMAC Cipher-Based Message Authentication Code

CTR Counter

CSPRNG Cryptographically Secure Pseudorandom Number Generator

DES Data Encryption Standard

DMA Direct Memory Access

DRAM Dynamic Random-Access Memory

FCK File Content Key

FDE Full Disk Encryption

GPT GUID Partition Table

HMAC Hash-Based Message Authentication Code

IV Initialization Vector

JSON JavaScript Object Notation

KDF Key Derivation Function

KEK Key Encryption Key

LUKS Linux Unified Key Setup

MAC Message authentication code

NIST National Institute of Standards and Technology

OS Operating System

PBE Pre-Boot Environment

PIN Personal Identification Number

PRNG Pseudorandom Number Generator

RAM Random Access Memory

ROM Read Only Memory

RFC Request For Comments

SIV Synthetic Initialization Vector

SSD Solid-State Drive

SSO Single Sing-On

UI User Interface

UID Unique Identifier

USB Universal Serial Bus

VDE Virtual Disk Encryption

VFS Virtual File System

Appendix **B**

Contents of enclosed SD

```
 readme.txt ........................ the file with SD contents description
├─ exe .................................... the directory with executables
├─ src ..................................... the directory of source codes
│  ├─ bp-cryptomator ........................... implementation sources
│  └─ thesis .............. the directory of LaTeX source codes of the thesis
└─ text .......................................... the thesis text directory
   └─ thesis.pdf ......................... the thesis text in PDF format
```