Security Target
for
Tresorit Core Interface v5.0

Version	v2.1
Date	2023-10-27
Classification	PUBLIC

Version history

Version	Date	Author	Description
v1.0	2022-05-19	Tresorit Kft.	The first version of the Security Target.
v1.1	2022-10-06	Tresorit Kft.	Improved version with additional clarifications.
v1.2	2022-12-08	Tresorit Kft.	Improved version with additional clarifications.
v1.3	2023-03-31	Tresorit Kft.	Improved version with additional clarifications. Version history format consistent with other documents.
v1.4	2023-06-23	Tresorit Kft.	Improved version with additional clarifications.
v2.0	2023-07-10	Tresorit Kft.	Improved version with additional clarifications.
v2.1	2023-10-27	Tresorit Kft.	Improved version with additional clarifications.

Table of Contents

1 Introduction

This section identifies the Security Target (ST) and the Target of Evaluation (TOE). The TOE is Tresorit Core Interface v5.0 and will be referred to as the TOE in this Security Target.

1.1 Security Target and TOE References

ST Title	Security Target for Tresorit Core Interface v5.0
ST Version	v2.1
ST Creation Date	2023-10-27
TOE Short Name	Tresorit Core Interface v5.0
TOE Reference	Tresorit Core Interface v5.0.3950.3950

Table 1.1 — Security Target and TOE References

1.2 TOE Overview

The TOE is a command line interface (CLI) application for Windows designed as an end-to-end encrypted file storage and sharing solution to protect the confidentiality and integrity of users' files and file names. Users can share their files and folders with other users of the TOE. User files and folders are accessible to those who the user gave access to.

Users of the TOE can create cryptographically protected containers and share entire containers or part of their content with other users of the TOE. The TOE provides an end-to-end encrypted solution which guarantees that no one who has access to the communication channel between the users can access or modify the content of the shared files or the name of the files. The TOE provides its security functionality without the need to trust the developer's servers.

The TOE implements a state-of-the-art end-to-end encryption (E2EE) communication that ensures all encrypted information remains encrypted once it leaves the sender's device and remains encrypted until it reaches the recipient. This means that no third party has any way of accessing the exchanged information in unencrypted form, not even the developer of the TOE. As an industry-standard best practice, the TOE also uses TLS 1.2+ for in-transit data encryption when communicating with the cloud servers, even though this is not part of the TOE's end-to-end encrypted security model and thus not part of the TOE's security requirements; the TOE would remain secure even without using any kind of in-transit data encryption.

1.2.1 TOE Usage and Major Security Features

The TOE can be used to share files and folders with other TOE users in a secure way that guarantees that no third party has any way of accessing the exchanged information in unencrypted form.

The TOE is responsible for the confidentiality and integrity protection of user file contents, file and folder names transmitted over network with state-of-the-art encryption solutions based on cryptographic best practices.

The major security features of the TOE are the following:

- **Encrypted communication with other TOE users:** The TOE encrypts the names and contents of all files leaving the TOE, meaning that no cloud servers or network devices have access to the file names and file content shared between TOE users.
- **Secure key exchange:** Exchanging encryption keys between TOE applications is done in a way that only the communicating parties gain access to the keys, ensuring no third party can access them in unencrypted form.
- **Key generation and management:** The keys used for encrypting information are generated by the sending party and are managed by the participating parties in a way that no third party can gain access to them in unencrypted form, not even temporarily.
- **End-point authentication:** All parties can be sure that the public keys belong to the desired party, and a potential attacker cannot inject their own public key to execute a man-in-the-middle attack.

1.2.2 TOE Type

The TOE is a CLI software product designed for end-to-end encrypted file/folder sharing.

1.2.3 Non-TOE Hardware/Software/Firmware

The TOE is a software product which requires the following OS to run:

- Windows: 10 (x64) or later

The TOE has the same minimum hardware requirements as the underlying OS. The TOE requires internet access and the usage of a pre-defined cloud storage service provided by the developer.

The cloud service is a Microsoft Azure infrastructure architected and operated by the developer of the TOE. The TOE automatically uses this service without any interaction required by the user.

1.3 TOE Description

This section addresses the physical and logical scope of the TOE.

Figure 1.1 describes the TOE and its environment. The two displayed TOEs are the running TOE software instances on the shown physical computers.

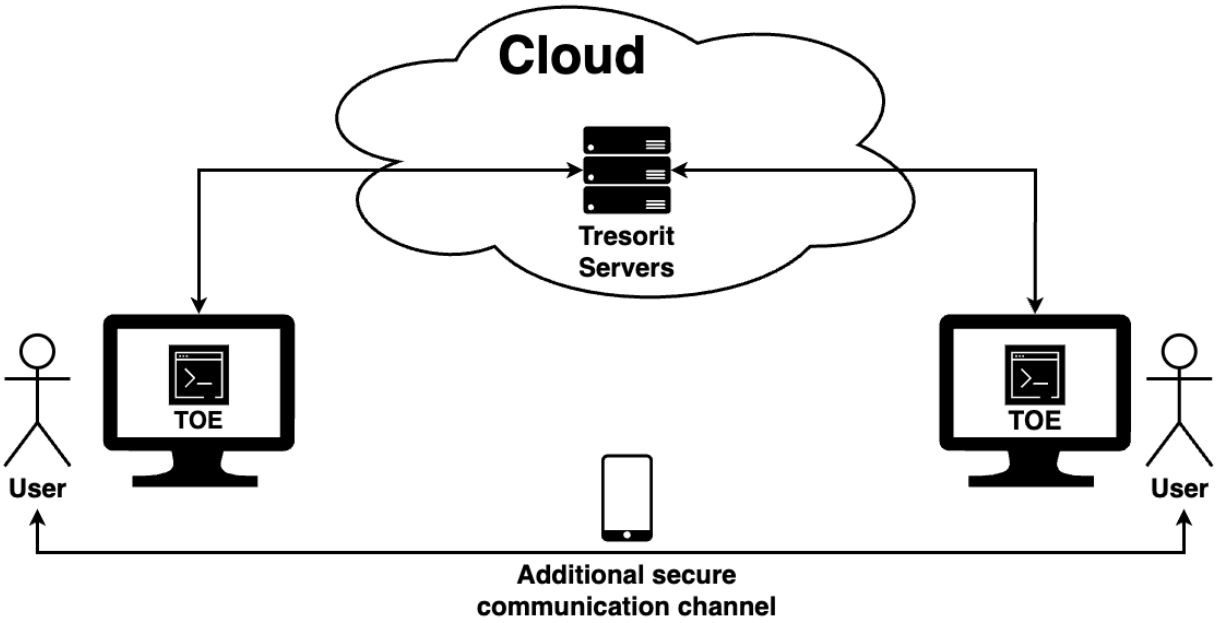

Figure 1.1 — The TOE and its environment

Figure 1.1 illustrates an additional secure communication channel. This channel is needed to be established when a TOE user (inviter) shares a container with another TOE user (invitee) to communicate verification objects required to ensure the integrity of the container sharing process and is only needed until the invitee accepts the invitation to the container.

1.3.1 Physical Scope of the TOE

The TOE is a CLI application delivered to the users in the form of an installer named Tresorit.exe with version 3.5.4549.3950.

The physical scope of the TOE consists of two files along with the necessary guidance documentation.

Table 1.2 details parts of the physical scope of the TOE.

File	Identification	Version
TOE executable	tresorit-core-interface.exe (the executable file which is to be started by the user to use the TOE)	5.0.3950.3950
TOE DLL	Tresorit.dll (the DLL file containing TOE functionality, used by the TOE executable)	4.13.17.3950
User guidance	AGD Documentation for Tresorit Core Interface v5.0 (PDF document, user guide and install guide)	1.1

Table 1.2 — The physical scope of the TOE

The TOE executable and the TOE DLL are delivered to the user in the form of an executable installer named Tresorit.exe.[1] Said installer installs two Tresorit products:

1. a GUI application — this is not part of the TOE,
2. a CLI application — this is the TOE executable (tresorit-core-interface.exe).

Both the GUI and the CLI application use the same TOE DLL (Tresorit.dll), which is also part of the TOE and contains functionality used by both applications. The TOE DLL is also installed by the installer.

The installer file has a separate version number than the TOE executable and the TOE DLL.

The CLI application uses a different set of functionalities than the GUI application. The available functions of the TOE are listed in Chapter 7 of this document.

Figure 1.2 illustrates the physical scope of the TOE software, including the TOE delivery method. The boxes with green background are parts of the TOE software, others are not.

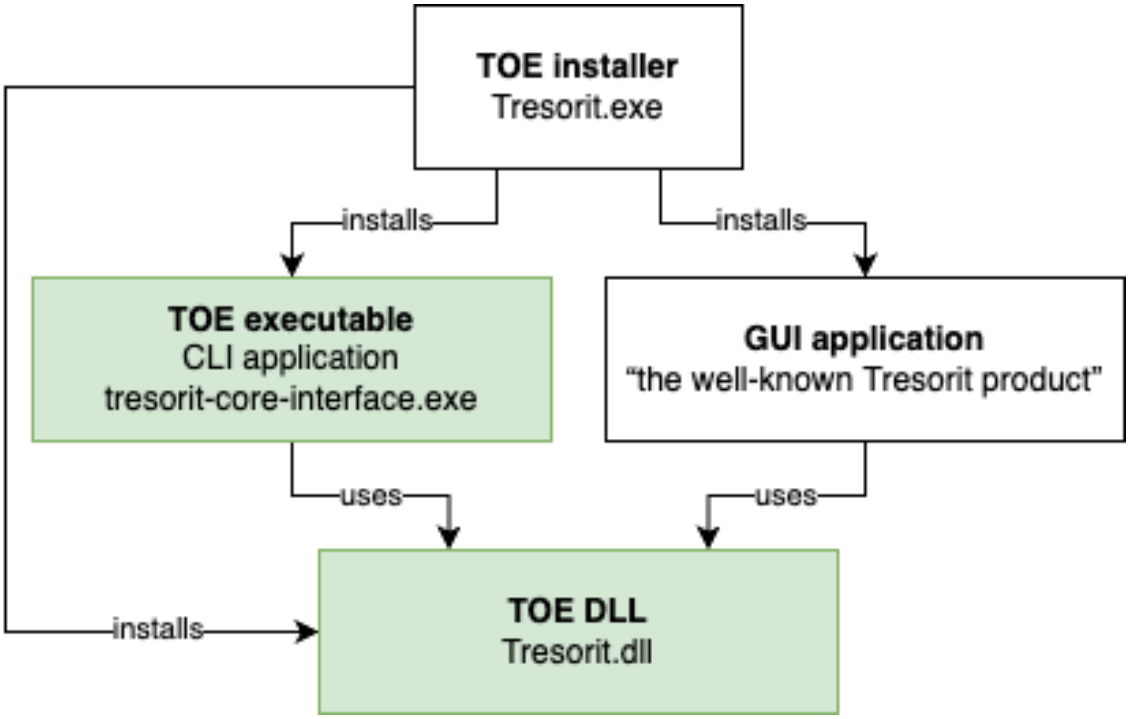

Figure 1.2 — The physical scope of the TOE software including the TOE delivery method

The developer maintains an email address for customers to ask their questions at "support@tresorit.com". The user guidance documentation and

[1] The installer is not considered to be part of the physical scope of the TOE, but a TOE delivery method.

the installer of the TOE can be accessed for users upon request submitted through this email address or through dedicated sales representatives for enterprise customers.

Each part of the TOE (the TOE executable, the TOE DLL and the guidance document) and the TOE installer are digitally signed by the developer.

1.3.2 Logical Scope of the TOE

The logical boundary of the TOE is broken down into the following security classes:

- Cryptographic operations
- File management
- Container management
- User management
- Secure communication

1.3.2.1 Cryptographic operations

This functionality is provided by the TOE for encryption and decryption of user files, as well as file and folder names. It is responsible for encrypted communication with the cloud service servers also. It performs cryptographic key generation, cryptographic key management, and cryptographic operations (e.g., hashing) as well.

1.3.2.2 File management

Files are organized in an arbitrary structure of folders and subfolders within a shared container. These files can be on the container level. The TOE encrypts all uploaded files and folders. Every file has a unique combination of a symmetric encryption key and a random initialization vector. As a result, even if two files only differ by one bit, their encrypted form will be completely different.

1.3.2.3 Container management

TOE users can create encrypted containers to upload files and folders into it and share them with other TOE users. The thumbprint of the container can be verified by the users to check the integrity of the shared data. Access to shared containers can be revoked.

1.3.2.4 User management

The TOE provides an interface to register, login, and logout, but the identification and authentication management is not handled by the TOE. TOE users can get other users' public keys as well.

1.3.2.5 Secure communication

The TOE encrypts files and folders when they are transferred between TOE users in a way that protects their confidentiality and integrity from third parties including the cloud servers used by the TOE.

1.3.3 Features and functionality not included in the TOE:

- Identity and authentication management
- Invitation of non-TOE users
- Link-based sharing
- Single Sign-On
- Enterprise features
- Web application features

1.4 Usage of the TOE in order to implement cryptographic controls

The security objectives of the TOE allow users to use the TOE to exchange data in an encrypted format in such a way that is described by international standards or required by legal regulations. The following table contains excerpts from a few international standards that require data to be encrypted. Using the TOE helps organizations to implement these controls easier.

Regulation / Standard	Excerpt
GDPR	Processing of personal data must be done in such a way as to ensure appropriate security, integrity, and confidentiality. The controller shall not be required to communicate the personal data breach to the data subject if encryption was applied to the personal data affected by the personal data breach, in particular if it renders the personal data unintelligible to any person who is not authorized to access it. Source: [GDPR].
HIPAA	Sensitive data shall be encrypted. Source: [HIPAA].
TISAX	The mapping of information classification (such as confidential or secret) to protection must be handled. Source: [TISAX].
BSI_C5	Authentication information shall be handled according to requirement PSS-07. Key management shall be handled according to requirement CRY-04. Access to cloud customer data shall be handled according to requirement IDM-07. Source: [BSI_C5].

CJIS	Access control mechanisms shall use encryption. Data transmitted outside the boundary of the physically secure location shall be encrypted. Source: [CJIS].
CMMC	Audit information shall be protected. Cryptographic mechanisms shall be used to protect digital media during transport. Confidentiality of backup shall be protected. Cryptographic mechanisms shall be implemented to prevent unauthorized disclosure of data during transmission. Source: [CMMC].
FEDRAMP	Cryptographic mechanisms shall be used to protect digital media during transport. Cryptographic mechanisms shall be implemented to prevent unauthorized disclosure of data during transmission. Cryptographic mechanisms shall be implemented to prevent unauthorized disclosure and modification of information. Source: [FEDRAMP].
FINRA	Sensitive data and/or storage units shall be encrypted. Source: [FINRA].
ISO27001	Documented information required by the information security management system shall be controlled to ensure it is adequately protected (e.g., from loss of confidentiality, improper use, or loss of integrity). Source: [ISO27001].
ITAR	Usage of end-to-end encrypted services transferring technical data shall not count as exports, reexports, retransfers, or temporary imports. Source: [ITAR].
NIST	Cryptographic mechanisms shall be implemented for security and privacy controls for Information Systems and Organizations according to [SP800-53].
SOC2	Encryption shall be used to protect data-at-rest. Encryption keys shall be protected. Encryption shall be used to protect transmission of data. Source: [SOC2].
DTL	Best practice cryptography shall be used to protect data-at-rest and data-at-transit. Source: [DTL].

2 Conformance Claims

Common Criteria Conformance	CC Part 2 Conformant, CC Part 3 Conformant

Common Criteria Version	Common Criteria for Information Technology Security Evaluation, Version 3.1, Revision 5, April 2017
PP Conformance	—
Evaluation Assurance Level	EAL4 augmented with AVA_VAN.5

2.1 Protection Profile Conformance Rationale

This ST does not claim conformance to a PP.

3 Security Problem Definition

This section includes the following:

- Assets
- Secure usage assumptions,
- Threats, and
- Organisational security policies.

3.1 Assets

Asset	Description
ASSET.DATA	Contents of data files that contain information to be protected and names of files and folders stored in TOE containers.
ASSET.PASSWORD	Password of the TOE user.
ASSET.USER_KEY	User's cryptographic keys that are transmitted to cloud servers in an encrypted format. These keys are only accessible for a single TOE user. Public keys are excluded because they are meant to be shared.
ASSET.CONTAINER_KEY	Container's cryptographic keys that are transmitted to cloud servers in an encrypted format and distributed among TOE users based on who the containers are shared with. Public keys are excluded because they are meant to be shared.

3.2 Assumptions

Assumption	Description
A.PLATFORM	The underlying operating system and hardware platform on which the TOE is installed are trustworthy, work correctly, and have no undocumented security critical side effects on the security functions of the TOE.

A.PHYSICAL	The TOE is operated on a hardware platform to which only the TOE user has physical access to.
A.ENTROPY	The underlying operating system on which the TOE is installed provides an entropy source that is suitable for generating cryptographically secure pseudorandom numbers.
A.USER	The TOE user is trustworthy, security conscious, and uses the TOE according to the provided user guidance.
A.PASSWORD	The password chosen by the TOE user for protecting ASSET.USER_KEY is kept secret.

3.3 Threats

Threat agents are attackers who have access to any communication channel over which the integrity protected, and confidentiality protected data are transferred, e.g., networks, other paths of transmission, the cloud service's storage media etc.

The threat agent is assumed to have advanced attack potential.

Threat	Description
T.DISCLOSE	Loss of confidentiality — An attacker of one of the communication paths over which the assets are transferred succeeds in accessing the assets, i.e., the attacker violates the confidentiality of the information sent over the communication channel.[2]
T.INJECT	Loss of integrity — An attacker of one of the communication paths over which the assets are transferred adds new data files (files that are not in the container and have never been in the container before) to a container, in a way that is not detected.[3]
T.TAMPER	Loss of integrity — An attacker of one of the communication paths over which the assets are transferred replaces or modifies the content of the data

[2] The attack can for example be achieved by eavesdropping, recording encrypted data during the transfer and deciphering the encrypted data.

[3] The attack can for example be achieved by interrupting the transfer, modifying the server's responses, injecting new file entries, then serving those files as if they belonged to the container.

	files or the name of the data file in a way that is not detected.[4]
T.BRUTEFORCE	An attacker tries to crack the password through trial and error.

4 Security Objectives

4.1 Security Objectives for the TOE

Objective	Description
OT.CONFIDENTIALITY	The TOE shall provide mechanisms that protect the information of a transmitted data such that its content's confidentiality is protected and only accessible by authorized users.
OT.INTEGRITY	The TOE shall provide mechanisms that detect if an attacker has tampered with a transmitted data file (replacing or modifying the content or the name of data files).
OT.PASSWORD_KEY	The TOE shall use a password stretching algorithm with parameters that protects encrypted objects and the token used for server authentication against known brute-force attacks.
OT.PASSWORD	The TOE shall ensure that users create strong password.

4.2 Security Objectives for the Operational Environment

Objective	Description
OE.PLATFORM	The underlying operating system and hardware platform on which the TOE is installed shall be trustworthy, work correctly, and have no undocumented security critical side effects on the security functions of the TOE.
OE.PHYSICAL	The TOE shall be run on a hardware platform to which only the TOE user has physical access to.

[4] The attack can for example be achieved by interrupting the transfer, modifying the content of the data (including replacing the whole file or modify file/folder names information) and then re-constructing the integrity protection. Afterwards the modified data is sent to the intended destination.

OE.USER	The TOE user shall be trustworthy, security conscious, and shall use the TOE according to the provided user guidance including the that the TOE password is kept secret.
OE.ENTROPY	The underlying operating system on which the TOE is installed shall provide an entropy source that is suitable for generating cryptographically secure pseudorandom numbers.

4.3 Security Objectives Rationale

This section demonstrates that the stated security objectives counter all identified threats, policies, or assumptions.

The following tables provide a mapping of security objectives to the environment defined by the threats, policies, and assumptions, illustrating that each security objective covers at least one threat, policy, or assumption and that each threat, policy or assumption is covered by at least one security objective.

4.3.1 Security Objectives and Threats

Threats	Security Objectives for the TOE and Rationale
T.DISCLOSE	OT.CONFIDENTIALITY ensures that assets are protected in a way that threat agents cannot violate their confidentiality.
T.INJECT	OT.INTEGRITY ensures that injection of new files is detected.
T.TAMPER	OT.INTEGRITY ensures that tampering with a transmitted data file is detected.
T.BRUTEFORCE	OT.PASSWORD_KEY ensures that the password key derived from the user password is strong enough that known brute-force attacks cannot succeed against it.
	OT.PASSWORD ensures that the user password used as a base for the password key derivation is strong enough to use it for password key derivation.
	OT.CONFIDENTIALITY ensures that assets are protected with a cryptographic algorithm that protects against brute-force attacks.

Threats	Security Objectives for the Environment and Rationale

T.DISCLOSE	OE.ENTROPY ensures that used cryptographic algorithms are working with random numbers that have enough entropy to mitigate cryptographic attacks against encrypted assets.
	OE.PLATFORM ensures that the underlying operating system and hardware platform on which the TOE is installed are trustworthy, work correctly and have no undocumented security critical side effects on the security functions of the TOE.
	OE.PHYSICAL ensures that TOE is operated on a hardware platform to which only the TOE user has physical access to.
	OE.USER ensures that the TOE user is trustworthy, security conscious, and uses the TOE according to the provided user guidance including the that the TOE password is kept secret.

4.3.2 Assumptions and Security Objectives

Assumptions	Security Objectives for the Environment and Rationale
A.PLATFORM	OE.PLATFORM ensures that the underlying operating system and hardware platform on which the TOE is installed is trustworthy, work correctly and have no undocumented security critical side effects on the security functions of the TOE.
A.PHYSICAL	OE.PHYSICAL ensures that the TOE is operated on a hardware platform to which only the TOE user has physical access to.
A.ENTROPY	OE.ENTROPY ensures that the underlying operating system on which the TOE is installed provides an entropy source that is suitable for generating cryptographically secure pseudorandom numbers.
A.USER	OE.USER ensures that the TOE user is trustworthy, security conscious, and uses the TOE according to the provided user guidance.
A.PASSWORD	OE.USER ensures that the TOE user uses the TOE according to the provided user guidance including that a strong password is created, and it is kept secret.

5 Extended Components Definition

There are no extended SFRs and no extended Security Assurance Requirements (SAR) for the TOE.

6 Security Requirements

6.1 Conventions

The following conventions are used whenever an operation (assignment, selection, or refinement) has been applied to a security functional requirement:

- Assignment: *Italicized text*
- Selection: Underlined text
- Refinement: **Bold text**
- Assignment in selection: *Italicized and underlined text*

Whenever a security functional requirement has been used more than once, the title of the security functional requirement is followed by a unique string (e.g., /xyz) to distinguish between the different iterations of the security functional requirement.

6.2 Subjects, Objects, Security Attributes, and Operations

Subject	Description
S.USER	User of the TOE.
S.CRYPTO	Cryptographic module of the TOE that performs cryptographic operations automatically or manually when triggered by user action.
S.COMMUNICATION	Communication module of the TOE that handle sending and receiving data to and from the cloud provider's servers (export from and import to the TOE).

Object	Description
O.DATA	Data file contents, file names, and folder names that contain information to be protected.
O.USER_PASSWORD_ DERIVED_TOKEN	A token is derived from the user's password during registration which is shared with the cloud servers. This token can be used by the servers to authenticate the TOE user during subsequent logins.

O.USER_PASSWORD_ DERIVED_KEY	A key that is derived from the user's password which is used to open encrypted user key objects.
O.USER_KEY	User's cryptographic keys that are transmitted to cloud servers in an encrypted format.
O.USER_VERIFICATION_ HASH	It is a hash which is used to verify the integrity of user's public keys.
O.USER_PUBLIC_KEY	Public key counterparts of asymmetrical O.USER_KEY objects.
O.CONTAINER_KEY	Container's cryptographic keys that are transmitted to cloud servers in an encrypted format and distributed among TOE users based on who the containers are shared with.
O.CONTAINER_ VERIFICATION_HASH	It is a hash that is used to verify the integrity of a container.

Security attribute	Description
SA.PASSWORD	The password of the TOE user.

6.2.1 Operations

6.2.1.1 OP.HASH_ACCESS

- Subject: S.USER
- Objects: O.USER_VERIFICATION_HASH, O.CONTAINER_VERIFICATION_HASH
- Description: S.USER accesses O.USER_VERIFICATION_HASH and O.CONTAINER_VERIFICATION_HASH objects to verify integrity.

6.2.1.2 OP.DOWNLOAD

- Subject: S.USER
- Objects: O.DATA
- Description: S.USER accesses O.DATA after download and decryption.

6.2.1.3 OP.UPLOAD

- Subject: S.USER
- Objects: O.DATA
- Description: S.USER can instruct the TOE to upload encrypted O.DATA to the cloud.

6.2.1.4 OP.PASSWORD_DERIVATION

- Subject: S.CRYPTO
- Objects: O.USER_PASSWORD_DERIVED_TOKEN, O.USER_PASSWORD_DERIVED_KEY
- Description: S.CRYPTO derives O.USER_PASSWORD_DERIVED_TOKEN and O.USER_PASSWORD_DERIVED_KEY from SA.PASSWORD that the user provided. (Note: Performed during registration and login.)

6.2.1.5 OP.KEY_CREATION

- Subject: S.CRYPTO
- Objects: O.USER_KEY, O.USER_PUBLIC_KEY, O.CONTAINER_KEY
- Description: S.CRYPTO generates or derives O.USER_KEY, O.USER_PUBLIC_KEY and O.CONTAINER_KEY objects.

6.2.1.6 OP.USER_KEY_ENCRYPTION

- Subject: S.CRYPTO
- Objects: O.USER_KEY, O.USER_PASSWORD_DERIVED_KEY, O.USER_PUBLIC_KEY
- Description: S.CRYPTO encrypts O.USER_KEY objects with O.USER_PASSWORD_DERIVED_KEY or with O.USER_PUBLIC_KEY or with other O.USER_KEY objects.

6.2.1.7 OP.USER_KEY_DECRYPTION

- Subject: S.CRYPTO
- Objects: O.USER_KEY, O.USER_PASSWORD_DERIVED_KEY
- Description: S.CRYPTO decrypts encrypted O.USER_KEY objects with O.USER_PASSWORD_DERIVED_KEY or with other O.USER_KEY objects.

6.2.1.8 OP.DATA_ENCRYPTION

- Subject: S.CRYPTO
- Objects: O.DATA, O.CONTAINER_KEY
- Description: S.CRYPTO encrypts O.DATA with O.CONTAINER_KEY objects.

6.2.1.9 OP.DATA_DECRYPTION

- Subject: S.CRYPTO
- Objects: O.DATA, O.CONTAINER_KEY
- Description: S.CRYPTO decrypts encrypted O.DATA with O.CONTAINER_KEY objects.

6.2.1.10 OP.HASH_GENERATION

- Subject: S.CRYPTO

- Objects: O.USER_VERIFICATION_HASH, O.CONTAINER_VERIFICATION_HASH
- Description: S.CRYPTO generates O.USER_VERIFICATION_HASH and O.CONTAINER_VERIFICATION_HASH objects.

6.2.1.11 OP.CONTAINER_KEY_ENCRYPTION

- Subject: S.CRYPTO
- Objects: O.CONTAINER_KEY, O.USER_PUBLIC_KEY
- Description: S.CRYPTO encrypts O.CONTAINER_KEY objects with O.USER_PUBLIC_KEY objects or with other O.CONTAINER_KEY objects.

6.2.1.12 OP.CONTAINER_KEY_DECRYPTION

- Subject: S.CRYPTO
- Objects: O.CONTAINER_KEY, O.USER_KEY
- Description: S.CRYPTO decrypts encrypted O.CONTAINER_KEY objects with O.USER_KEY objects or with other O.CONTAINER_KEY objects.

6.2.1.13 OP.EXPORT

- Subject: S.COMMUNICATION
- Objects: O.DATA, O.CONTAINER_KEY, O.USER_KEY, O.USER_PASSWORD_DERIVED_TOKEN, O.USER_PUBLIC_KEY, O.USER_PASSWORD_DERIVED_KEY
- Description: S.COMMUNICATION exports encrypted O.DATA, encrypted O.CONTAINER_KEY, encrypted O.USER_KEY to cloud servers. S.COMMUNICATION exports O.USER_PASSWORD_DERIVED_TOKEN, O.USER_PUBLIC_KEY to cloud servers. S.COMMUNICATION never exports SA.PASSWORD or O.USER_PASSWORD_DERIVED_KEY.

6.2.1.14 OP.IMPORT

- Subject: S.COMMUNICATION
- Objects: O.DATA, O.CONTAINER_KEY, O.USER_KEY, O.USER_PUBLIC_KEY
- Description: S.COMMUNICATION imports encrypted O.DATA, encrypted O.CONTAINER_KEY, and encrypted O.USER_KEY from cloud servers. S.COMMUNICATION imports O.USER_PUBLIC_KEY from cloud servers.

6.3 TOE Security Functional Requirements

6.3.1 Security functional policies implemented by the TOE

6.3.1.1 Data access

The cloud provider for the TOE stores the contents of data files and the names of files/directories in an encrypted format. This data is always encrypted using the containers' cryptographic keys before uploading it to the cloud servers. The user must enter the correct password during login every time after starting the TOE or after a logout to access the unencrypted data.The TOE imports objects automatically from the cloud servers and when a TOE user instructs it, manually. When the TOE imports this data, it decrypts it using the same keys and always checks its integrity. The TOE user can only access this data after successful decryption and integrity verification.

Every file and directory are attached to a container. This association is exported when the TOE uploads encrypted data to the cloud servers.

6.3.1.2 Key derivation from password

A key is derived from the user’s password during registration and login with a password stretching algorithm. This SFP regulates that the user must choose a password when registering, and provide the same password every time when logging in. Neither the user’s password nor this derived key can be exported to outside of the TOE boundary, not even in an encrypted format. (Note: Another output, a token is also derived from the user’s password during registration and login, also with a password stretching algorithm. This token is not considered secret or sensitive data and can be exported from the TOE without restrictions. This token is shared with the cloud servers during registration. During login, this token is used to prove to the cloud servers that the user knows the password.)

6.3.1.3 User key access

The cloud provider for the TOE stores cryptographic user keys in an encrypted format. These keys are always encrypted using other user keys of the same TOE user or using the key which is derived from the user’s password before uploading it to the cloud servers. The user must enter the correct password during login every time after starting the TOE or after a logout to access these keys. The TOE imports user keys automatically from the cloud servers. When the TOE imports these keys, it decrypts it using the same keys as for the encryption and always checks their integrity. The TOE only uses these keys to encrypt or decrypt other data after successful decryption and integrity verification.

Every user key is attached to a user. This association is exported when the TOE uploads encrypted user keys to the cloud servers.

Each user key that belongs to an asymmetric key pair is either a public key or a private key. User keys that are public keys of such asymmetric key pairs are not considered secret or sensitive data, and can be exported from the TOE without restrictions.

6.3.1.4 Container key access

The cloud provider for the TOE stores cryptographic container keys in an encrypted format. These keys are always encrypted using other container keys of the same container or using user public keys of the users who have been given access to that container. The user must enter the correct password during login every time after starting the TOE or after a logout to access these keys. The TOE imports user keys automatically from the cloud servers. When the TOE imports these keys, it decrypts them using the same keys that were used for encryption (the corresponding user private keys are used for decryption when user public keys were used for encryption), checking their integrity in the process. The TOE only uses these keys to encrypt or decrypt other data after successful decryption and integrity verification.

Integrity verification is performed automatically or manually. In most cases the TOE checks the integrity of the received keys automatically. If that is not possible, the TOE user needs to acquire a verification object (e.g., hash) over a secondary communication channel with another TOE user. The other TOE user can use their own TOE to generate these verification objects.

Manual integrity verification is required in two cases:

- When a TOE user is sharing a container with another TOE user using the other user's public key, the other TOE user must acquire a verification object of their own user key.
- When a TOE user receives an invitation to a container from another TOE user, the other TOE user must acquire a verification object of the container.

Each container key is attached to a container – this association is exported when the TOE uploads encrypted container keys to the cloud servers. Each container has an associated list of users and user public keys which can access that container – this list is exported when the TOE uploads container keys to the cloud servers.

6.3.2 Security Functional Requirements

6.3.2.1 FCS_CKM.1/rsa Cryptographic key generation

FCS_CKM.1.1 The TSF shall generate cryptographic keys in accordance with a specified cryptographic key generation algorithm *RSA Key-Pair Generation*

with a Fixed Public Exponent[5] and specified cryptographic key sizes *4096 bits*[6] that meet the following: *[SP800-56Br2]*[7].

6.3.2.2 FCS_CKM.1/random Cryptographic key generation

FCS_CKM.1.1 The TSF shall generate cryptographic keys in accordance with a specified cryptographic key generation algorithm *Using the Output of a Random Bit Generator*[8] and specified cryptographic key sizes *256 bits for AES and HMAC*[9] that meet the following: *[SP800-133r2]*[10].

6.3.2.3 FCS_CKM.1/password_stretching Cryptographic key generation

FCS_CKM.1.1 The TSF shall generate cryptographic keys in accordance with a specified cryptographic key generation algorithm *Scrypt with a cost parameter of 32768 or 131072, a block size of 8 and a parallelization parameter of 1 or 8, using user input or another cryptographic key as its input*[11] and specified cryptographic key sizes *256 bits*[12] that meet the following: *[RFC 7914]*[13].

6.3.2.4 FCS_CKM.1/derivation Cryptographic key generation

FCS_CKM.1.1 The TSF shall generate cryptographic keys in accordance with a specified cryptographic key generation algorithm *using the output of a key derivation (FCS_COP.1/key_derivation) or keyed hash (FCS_COP.1/keyed_hash), with inputs that include at least one other cryptographic key that is generated using one of the FCS_CKM.1/random, FCS_CKM.1/password_stretching, or FCS_CKM.1/derivation*[14] and specified cryptographic key sizes *256 or 512 bits*[15] that meet the following: *[RFC8018] (FCS_COP.1/key_derivation), [FIPS198-1] (FCS_COP.1/keyed_hash)*[16].

6.3.2.5 FCS_COP.1/hash Cryptographic operation

FCS_COP.1.1 The TSF shall perform *hashing and integrity verification*[17] in accordance with a specified cryptographic algorithm *SHA-2 with a digest*

---

[5] [assignment: cryptographic key generation algorithm]
[6] [assignment: cryptographic key sizes]
[7] [assignment: list of standards]
[8] [assignment: cryptographic key generation algorithm]
[9] [assignment: cryptographic key sizes]
[10] [assignment: list of standards]
[11] [assignment: cryptographic key generation algorithm]
[12] [assignment: cryptographic key sizes]
[13] [assignment: list of standards]
[14] [assignment: cryptographic key generation algorithm]
[15] [assignment: cryptographic key sizes]
[16] [assignment: list of standards]
[17] [assignment: list of cryptographic operations]

size of 256 bits[18] and cryptographic key sizes *none*[19] that meet the following: *[FIPS180-4]*[20].

6.3.2.6 FCS_COP.1/keyed_hash Cryptographic operation

FCS_COP.1.1 The TSF shall perform *keyed hashing and integrity verification*[21] in accordance with a specified cryptographic algorithm *HMAC with SHA-2 with a digest size of 256 or 512 bits*[22] and cryptographic key sizes *256 or 512 bits*[23] that meet the following: *[FIPS198-1]*[24].

6.3.2.7 FCS_COP.1/key_derivation Cryptographic operation

FCS_COP.1.1 The TSF shall perform *key derivation*[25] in accordance with a specified cryptographic algorithm *PBKDF2 with HMAC with SHA-2 with a digest size of 256 or 512 bits*[26] and cryptographic key sizes *256 or 512 bits*[27] that meet the following: *[RFC8018]*[28].

6.3.2.8 FCS_COP.1/enc_symmetric_noauth Cryptographic operation

FCS_COP.1.1 The TSF shall perform *encryption and decryption*[29] in accordance with a specified cryptographic algorithm *AES with ECB or CFB mode of operation*[30] and cryptographic key sizes *256 bits*[31] that meet the following: *[SP800-38A]*[32].

6.3.2.9 FCS_COP.1/enc_symmetric_auth Cryptographic operation

FCS_COP.1.1 The TSF shall perform *encryption, decryption, and integrity verification*[33] in accordance with a specified cryptographic algorithm *AES with GCM mode of operation, a tag size of 128 bits, and an initialization vector size of 128 bits*[34] and cryptographic key sizes *256 bits*[35] that meet the following: *[SP800-38D]*[36].

---

[18] [assignment: cryptographic algorithm]
[19] [assignment: cryptographic key sizes]
[20] [assignment: list of standards]
[21] [assignment: list of cryptographic operations]
[22] [assignment: cryptographic algorithm]
[23] [assignment: cryptographic key sizes]
[24] [assignment: list of standards]
[25] [assignment: list of cryptographic operations]
[26] [assignment: cryptographic algorithm]
[27] [assignment: cryptographic key sizes]
[28] [assignment: list of standards]
[29] [assignment: list of cryptographic operations]
[30] [assignment: cryptographic algorithm]
[31] [assignment: cryptographic key sizes]
[32] [assignment: list of standards]
[33] [assignment: list of cryptographic operations]
[34] [assignment: cryptographic algorithm]
[35] [assignment: cryptographic key sizes]
[36] [assignment: list of standards]

6.3.2.10 FCS_COP.1/enc_asymmetric Cryptographic operation

FCS_COP.1.1 The TSF shall perform *encryption and decryption*[37] in accordance with a specified cryptographic algorithm *RSA-OAEP with hash function SHA-1*[38] and cryptographic key sizes *4096 bits*[39] that meet the following: *[SP800-56Br2]*[40].

6.3.2.11 FDP_ACC.1 Subset access control

FDP_ACC.1.1 The TSF shall enforce the *following SFPs*[41] on

- *Data access SFP*
  - *Subjects:*
    - *S.USER*
    - *S.CRYPTO*
    - *S.COMMUNICATION*
  - *Objects:*
    - *O.DATA*
    - *O.CONTAINER_KEY*
  - *Operations:*
    - *OP.DOWNLOAD*
    - *OP.UPLOAD*
    - *OP.DATA_ENCRYPTION*
    - *OP.DATA_DECRYPTION*
    - *OP.EXPORT*
    - *OP.IMPORT*
- *User key access SFP*
  - *Subjects:*
    - *S.CRYPTO*
    - *S.COMMUNICATION*
  - *Objects:*
    - *O.USER_KEY*
    - *O.USER_PUBLIC_KEY*
    - *O.USER_PASSWORD_DERIVED_KEY*
  - *Operations:*
    - *OP.KEY_CREATION*
    - *OP.USER_KEY_ENCRYPTION*
    - *OP.USER_KEY_DECRYPTION*
    - *OP.EXPORT*
    - *OP.IMPORT*
- *Container key access SFP*

---

[37] [assignment: list of cryptographic operations]
[38] [assignment: cryptographic algorithm]
[39] [assignment: cryptographic key sizes]
[40] [assignment: list of standards]
[41] [assignment: access control SFP]

    - *Subjects:*
        - *S.USER*
        - *S.CRYPTO*
        - *S.COMMUNICATION*
    - *Objects:*
        - *O.CONTAINER_KEY*
        - *O.USER_KEY*
        - *O.USER_PUBLIC_KEY*
        - *O.CONTAINER_VERIFICATION_HASH*
        - *O.USER_VERIFICATION_HASH*
    - *Operations:*
        - *OP.HASH_ACCESS*
        - *OP.KEY_CREATION*
        - *OP.HASH_GENERATION*
        - *OP.CONTAINER_KEY_ENCRYPTION*
        - *OP.CONTAINER_KEY_DECRYPTION*
        - *OP.EXPORT*
        - *OP.IMPORT*
- *Key derivation from password SFP*
    - *Subjects:*
        - *S.CRYPTO*
        - *S.COMMMUNICATION*
    - *Objects:*
        - *O.USER_PASSWORD_DERIVED_KEY*
    - *Operations:*
        - *OP.PASSWORD_DERIVATION*
        - *OP.EXPORT*[42].

6.3.2.12 FDP_ACF.1 Security attribute based access control

FDP_ACF.1.1 The TSF shall enforce the *following SFPs*[43] to objects based on the following:

- *Data access SFP*
    - *Subjects:*
        - *S.USER*
        - *S.CRYPTO*
        - *S.COMMUNICATION*
    - *Objects:*
        - *O.DATA*
        - *O.CONTAINER_KEY*
    - *Security attributes:*

[42] [assignment: list of subjects, objects, and operations among subjects and objects covered by the SFP]
[43] [assignment: access control SFP]

        - *SA.PASSWORD*
- *User key access SFP*
    - *Subjects:*
        - *S.CRYPTO*
        - *S.COMMUNICATION*
    - *Objects:*
        - *O.USER_KEY*
        - *O.USER_PUBLIC_KEY*
        - *O.USER_PASSWORD_DERIVED_KEY*
    - *Security attributes:*
        - *SA.PASSWORD*
- *Container key access SFP*
    - *Subjects:*
        - *S.USER*
        - *S.CRYPTO*
        - *S.COMMUNICATION*
    - *Objects:*
        - *O.CONTAINER_KEY*
        - *O.USER_KEY*
        - *O.USER_PUBLIC_KEY*
        - *O.CONTAINER_VERIFICATION_HASH*
        - *O.USER_VERIFICATION_HASH*
    - *Security attributes:*
        - *SA.PASSWORD*
- *Key derivation from password SFP*
    - *Subjects:*
        - *S.CRYPTO*
        - *S.COMMMUNICATION*
    - *Objects:*
        - *O.USER_PASSWORD_DERIVED_KEY*
    - *Security attributes:*
        - *SA.PASSWORD*[44]

FDP_ACF.1.2 The TSF shall enforce the following rules to determine if an operation among controlled subjects and controlled objects is allowed:

- *The user must choose a SA.PASSWORD that meet the following password requirements which are based on Unicode code points and character classes:*
    - *Minimum of 8 characters long,*
    - *Contains at least 1 uppercase letter,*

[44] [assignment: list of subjects and objects controlled under the indicated SFP, and for each, the SFP-relevant security attributes, or named groups of SFP-relevant security attributes]

    - *Contains at least 1 lowercase letter,*
    - *Contains at least 1 digit*
- *The user must enter the correct SA.PASSWORD to access user data*[45].

FDP_ACF.1.3 The TSF shall explicitly authorise access of subjects to objects based on the following additional rules:

- *S.USER can access O.DATA when objects are already decrypted by S.CRYPTO.*
- *S.CRYPTO can always access the encrypted form of O.DATA, O.USER_KEY, O.CONTAINER_KEY.*
- *S.CRYPTO can only access the decrypted form of O.DATA, O.USER_KEY, O.CONTAINER_KEY if it was able to successfully decrypt said objects with their encryption keys.*
- *S.COMMUNICATION can access O.DATA, O.USER_KEY, O.CONTAINER_KEY in their encrypted form only*[46].

FDP_ACF.1.4 The TSF shall explicitly deny access of subjects to objects based on the following additional rules: *none*[47].

6.3.2.13 FDP_ETC.1 Export of user data without security attributes

FDP_ETC.1.1 The TSF shall enforce the *Data access SFP, User key access SFP, Container key access SFP, and Key derivation from password SFP*[48] when exporting user data, controlled under the SFP(s), outside of the TOE.

FDP_ETC.1.2 The TSF shall export the user data without the user data's associated security attributes.

6.3.2.14 FDP_ITC.1 Import of user data without security attributes

FDP_ITC.1.1 The TSF shall enforce the *Data access SFP, User key access SFP, and Container key access SFP*[49] when importing user data, controlled under the SFP, from outside of the TOE.

FDP_ITC.1.2 The TSF shall ignore any security attributes associated with the user data when imported from outside the TOE.

FDP_ITC.1.3 The TSF shall enforce the following rules when importing user data controlled under the SFP from outside the TOE: *none*[50].

---

[45] [assignment: rules governing access among controlled subjects and controlled objects using controlled operations on controlled objects]
[46] [assignment: rules, based on security attributes, that explicitly authorise access of subjects to objects]
[47] [assignment: rules, based on security attributes, that explicitly deny access of subjects to objects]
[48] [assignment: access control SFP(s) and/or information flow control SFP(s)]
[49] [assignment: access control SFP(s) and/or information flow control SFP(s)]
[50] [assignment: additional importation control rules]

6.3.2.15 FDP_DAU.1 Basic Data Authentication

FDP_DAU.1.1 The TSF shall provide a capability to generate evidence that can be used as a guarantee of the validity of

- *O.DATA*
- *O.USER_KEY*
- *O.USER_PUBLIC_KEY*
- *O.CONTAINER_KEY*[51].

FDP_DAU.1.2 The TSF shall provide

- *S.USER*
- *S.CRYPTO*[52]

with the ability to verify evidence of the validity of the indicated information.

Application Note FDP_DAU.1: S.USER's assistance is only needed for:

- verifying other users' public keys when inviting them to containers (using O.USER_VERIFICATION_HASH objects)
- accepting invitations to containers sent by other users (using O.CONTAINER_VERIFICATION_HASH objects)

All other evidence verification is handled by S.CRYPTO automatically.

6.3.2.16 FIA_SOS.2 TSF Generation of secrets

FIA_SOS.2.1 The TSF shall provide a mechanism to generate secrets that meet *[SP800-90Ar1]*[53].

FIA_SOS.2.2 The TSF shall be able to enforce the use of TSF generated secrets for

- *the initialization vector of block ciphers used for symmetric encryption (FCS_COP.1/enc_symmetric_auth and FCS_COP.1/enc_symmetric_noauth)*
- *the salt parameter of password stretching functions (FCS_CKM.1/password_stretching*)
- *the random salt used to derive keys if such values are required (FCS_CKM.1/derivation)*[54].

[51] [assignment: list of objects or information types]
[52] [assignment: list of subjects]
[53] [assignment: a defined quality metric]
[54] [assignment: list of TSF functions]

6.4 TOE Security Assurance Requirements

This section defines the assurance requirements for the TOE. Assurance requirements are taken from CC Part 3 and are EAL4+ augmented with AVA_VAN.5

Assurance requirements		
Class ASE: Security Target evaluation	ASE_CCL.1	Conformance claims
	ASE_ECD.1	Extended components definition
	ASE_INT.1	ST introduction
	ASE_OBJ.2	Security objectives
	ASE_REQ.2	Derived security requirements
	ASE_SPD.1	Security problem definition
	ASE_TSS.1	TOE summary specification
Class ALC: Life Cycle Support	ALC_CMC.4	Production support, acceptance procedures and automation
	ALC_CMS.4	Problem tracking CM Coverage
	ALC_DEL.1	Delivery procedures
	ALC_DVS.1	Identification of security measures
	ALC_LCD.1	Developer defined life-cycle model
	ALC_TAT.1	Well-defined development tools
Class ADV: Development	ADV_ARC.1	Security architecture description
	ADV_FSP.4	Complete functional specification
	ADV_IMP.1	Implementation representation of the TSF
	ADV_TDS.3	Basic modular design
Class AGD: Guidance documents	AGD_OPE.1	Operational user guidance
	AGD_PRE.1	Preparative procedures
Class ATE: Tests	ATE_COV.2	Analysis of coverage
	ATE_DPT.1	Testing: basic design
	ATE_FUN.1	Functional testing
	ATE_IND.2	Independent testing – sample
Class AVA: Vulnerability assessment	AVA_VAN.5	Advanced methodical vulnerability analysis

The developer has chosen EAL4+ because it is best suited to address the stated security objectives. EAL4+ challenges vendors to use best (rather

than average) security and commercial practices. EAL4+ allows the vendor to evaluate their product at a detailed level. The chosen assurance level is appropriate for the threats defined in the environment.

The augmentation of AVA_VAN.5 was chosen to give greater assurance of the product's security and the absence of vulnerabilities. At EAL4+ augmented with AVA_VAN.5 penetration testing is performed by the evaluator assuming an attack potential of Advanced.

6.5 Security Requirements Rationale

This section provides the rationale for necessity and sufficiency of security requirements, demonstrating that each of the security objectives is addressed by at least one security requirement, and that every security functional requirement is directed toward solving at least one objective.

6.5.1 Security Requirements Coverage and Sufficiency

The following table provides a mapping of the relationships of security requirements to objectives, illustrating that each security requirement covers at least one objective and that each objective is covered by at least one security requirement. The table in this section addresses the mapping of security functional requirements to security objectives.

Security Functional Requirements Related to Security Objectives

Objective	Functional Requirement and Rationale
OT.CONFIDENTIALITY	FCS_CKM.1/rsa ensures that RSA private keys are generated in a way that is considered secure and that there is no known attack against them which can be successful in recovering them from their corresponding public keys or the output of the cryptographic functions in which they are used as keys.
	FCS_CKM.1/random ensures that industry standard random bit generator is used for generating cryptographic keys.
	FCS_CKM.1/password_stretching ensures that password stretching is applied with secure parameters for protection against brute force attacks.
	FCS_CKM.1/derivation ensures that derivation operations are performed according to industry standard algorithms with secure parameters.
	FCS_COP.1/key_derivation ensures that key derivation operations are performed according to

	industry standard algorithms with secure parameters.
	FCS_COP.1/enc_symmetric_noauth ensures that symmetric encryption operations without authentication method are performed according to industry standard algorithms with secure parameters.
	FCS_COP.1/enc_symmetric_auth ensures that symmetric encryption operations with authentication operations are performed according to industry standard algorithms with secure parameters.
	FCS_COP.1/enc_asymmetric ensures that asymmetric encryption operations are performed according to industry standard algorithms with secure parameters.
	FDP_ACC.1 ensures that proper access control flow is applied to sensitive objects.
	FDP_ACF.1 ensures that access control is enforced on sensitive objects.
	FDP_ETC.1 ensures that user data is exported without security attributes.
	FIA_SOS.2 ensures that nonce and salt generation operations are performed according to industry standard algorithms with secure parameters
OT.INTEGRITY	FCS_COP.1/hash ensures that hashing operations are performed according to industry standard algorithms with secure parameters.
	FCS_COP.1/keyed_hash ensures that keyed hashing operations are performed according to industry standard algorithms with secure parameters.
	FDP_ACC.1 ensures that proper access control flow is applied to sensitive objects.
	FDP_ACF.1 ensures that access control is enforced on sensitive objects.
	FDP_ETC.1 ensures that user data is exported without security attributes.
	FDP_ITC.1 ensures that user data is imported without security attributes.

	FDP_DAU.1 ensures that guarantee of the validity of objects can be generated.
OT.PASSWORD_KEY	FCS_CKM.1/password_stretching ensures that password stretching is applied with secure parameters for protection against brute-force attacks.
	FDP_ACC.1 ensures that proper access control flow is applied to sensitive objects.
	FDP_ACF.1 ensures that access control is enforced on sensitive objects.
OT.PASSWORD	FDP_ACF.1 ensures that the user chooses a strong password.

6.6 Requirements Dependency Rationale

6.6.1 Rationale Showing that Dependencies are Satisfied

The selected security requirements include related dependencies. All SFR dependencies are satisfied except where FCS_CKM.4 or FMT_MSA.3 is a dependency.

FCS_CKM.4 is excluded per the following rationale. The TOE, being a compiled binary software, does not store cryptographic keys in itself — cryptographic keys in their unencrypted form are only ever stored in the RAM of the hardware running the TOE (and never in persistent storage). The hardware running the TOE is considered secure and inaccessible to attackers per OE.PLATFORM and OE.PHYSICAL, which makes key destruction on the hardware running the TOE irrelevant. The destruction of cryptographic keys stored outside the hardware running the TOE is also irrelevant because the TOE only exports cryptographic keys in an encrypted form. Hence, said keys cannot be recovered without their respective encryption keys, which are either only accessible by the running TOE instance (because they are stored in the RAM of the hardware running the TOE while the TOE is running) or can ultimately only be accessed with the user's password.

FMT_MSA.3 is excluded because the TSF does not provide default values for relevant object security attributes, which can be overridden by an initial value. The TOE does not have a default password, user passwords are created by users during registration.

6.6.1.1 Security Functional Requirements Dependencies

The following table provides a summary of the security functional requirements dependency analysis.

Component	Dependency	Rationale
FCS_CKM.1/rsa	FCS_CKM.4	Not included - See rationale above.
FCS_CKM.1/rsa	FCS_COP.1/enc_asymmetric	Included
FCS_CKM.1/random	FCS_CKM.4	Not included - See rationale above.
FCS_CKM.1/random	FCS_COP.1/keyed_hash FCS_COP.1/key_derivation FCS_COP.1/enc_symmetric_noauth FCS_COP.1/enc_symmetric_auth	Included
FCS_CKM.1/password_stretching	FCS_CKM.4	Not included - See rationale above.
FCS_CKM.1/password_stretching	FCS_COP.1/keyed_hash FCS_COP.1/key_derivation FCS_COP.1/enc_symmetric_noauth FCS_COP.1/enc_symmetric_auth	Included
FCS_CKM.1/derivation	FCS_CKM.4	Not included - See rationale above.
FCS_CKM.1/derivation	FCS_COP.1/keyed_hash FCS_COP.1/key_derivation FCS_COP.1/enc_symmetric_noauth FCS_COP.1/enc_symmetric_auth	Included
FCS_COP.1/hash	FCS_CKM.1	Not included - Hashing

		does not use cryptographic keys.
FCS_COP.1/hash	FCS_CKM.4	Not included - Hashing does not use cryptographic keys.
FCS_COP.1/keyed_hash	FCS_CKM.1/random FCS_CKM.1/password_stretching FCS_CKM.1/derivation	Included
FCS_COP.1/keyed_hash	FCS_CKM.4	Not included - See rationale above.
FCS_COP.1/key_derivation	FCS_CKM.1/random FCS_CKM.1/password_stretching FCS_CKM.1/derivation	Included
FCS_COP.1/key_derivation	FCS_CKM.4	Not included - See rationale above.
FCS_COP.1/enc_symmetric_noauth	FCS_CKM.1/random FCS_CKM.1/password_stretching FCS_CKM.1/derivation	Included
FCS_COP.1/enc_symmetric_noauth	FCS_CKM.4	Not included - See rationale above.
FCS_COP.1/enc_symmetric_auth	FCS_CKM.1/random FCS_CKM.1/password_stretching FCS_CKM.1/derivation	Included
FCS_COP.1/enc_symmetric_a	FCS_CKM.4	Not

uth		included - See rationale above.
FCS_COP.1/enc_asymmetric	FCS_CKM.1/rsa	Included
FCS_COP.1/enc_asymmetric	FCS_CKM.4	Not included - See rationale above.
FDP_ACC.1	FDP_ACF.1	Included
FDP_ACF.1	FDP_ACC.1	Included
	FMT_MSA.3	Not included – The TSF does not provide static attribute initialization.
FDP_ETC.1	FDP_ACC.1	Included
FDP_ITC.1	FDP_ACC.1	Included
	FMT_MSA.3	Not included – The TSF does not provide static attribute initialization.
FDP_DAU.1	-	-
FIA_SOS.2	-	-

6.6.1.2 Security Assurance Requirements Dependencies

Component	Depends On	Which is
ADV_ARC.1	ADV_FSP.1	included (hierarchical to ADV_FSP.4)
	ADV_TDS.1	included (hierarchical to ADV_TDS.3)

ADV_FSP.4	ADV_TDS.1	included (hierarchical to ADV_TDS.3)
ADV_IMP.1	ADV_TDS.3	included
	ALC_TAT.1	included
ADV_TDS.3	ADV_FSP.4	included
AGD_OPE.1	ADV_FSP.1	included (hierarchical to ADV_FSP.4)
AGD_PRE.1	-	not applicable
ALC_CMC.4	ALC_CMS.1	included (hierarchical to ALC_CMS.4)
	ALC_DVS.1	included
	ALC_LCD.1	included
ALC_CMS.4	-	not applicable
ALC_DEL.1	-	not applicable
ALC_DVS.1	-	not applicable
ALC_LCD.1	-	not applicable
ALC_TAT.1	ADV_IMP.1	included
ASE_INT.1	-	not applicable
ASE_CCL.1	ASE_INT.1	included or hierarchical component is included
	ASE_ECD.1	
	ASE_REQ.1	
ASE_SPD.1	-	not applicable
ASE_OBJ.2	ASE_SPD.1	included
ASE_ECD.1	-	not applicable
ASE_REQ.2	ASE_OBJ.2	included
	ASE_ECD.1	
ASE_TSS.1	ASE_INT.1	included or hierarchical component is included
	ASE_REQ.1	
	ADV_FSP.1	
ATE_COV.2	ADV_FSP.2	included (hierarchical to ADV_FSP.4)
	ATE_FUN.1	included
ATE_DPT.1	ADV_ARC.1	included
	ADV_TDS.2	included (hierarchical to ADV_TDS.3)

	ATE_FUN.1	included
ATE_FUN.1	ATE_COV.1	included (hierarchical to ATE_COV.2)
ATE_IND.2	ADV_FSP.2	included (hierarchical to ADV_FSP.4)
	AGD_OPE.1	included
	AGD_PRE.1	included
	ATE_COV.1	included (hierarchical to ATE_COV.2)
	ATE_FUN.1	included
AVA_VAN.5	ADV_ARC.1	included
	ADV_FSP.4	included
	ADV_TDS.3	included
	ADV_IMP.1	included
	AGD_OPE.1	included
	AGD_PRE.1	included
	ATE_DPT.1	included

7 TOE Summary Specification

This section provides a description of how the TOE satisfies all SFRs listed in this ST. The following table describes the mapping between security functionality and their associated SFRs:

Security functionality	SFRs
Cryptographic operations	FCS_CKM.1/rsa FCS_CKM.1/random FCS_CKM.1/password_stretching FCS_CKM.1/derivation FCS_COP.1/hash FCS_COP.1/keyed_hash FCS_COP.1/key_derivation FCS_COP.1/enc_symmetric_noauth FCS_COP.1/enc_symmetric_auth FCS_COP.1/enc_asymmetric FDP_DAU.1 FIA_SOS.2
File management	FDP_ACC.1 FDP_ACF.1 FDP_ETC.1 FDP_ITC.1 FDP_DAU.1
Container management	FDP_ACC.1 FDP_ACF.1 FDP_ETC.1 FDP_ITC.1 FDP_DAU.1
User management	FDP_ACC.1 FDP_ACF.1 FDP_ETC.1 FDP_ITC.1 FDP_DAU.1
Secure communication	FDP_ETC.1 FDP_ITC.1

7.1 Cryptographic operations

The TOE uses OpenSSL v3.1.4 for its cryptographic operations. OpenSSL is a software - a robust, commercial-grade, full-featured toolkit for general-purpose cryptography and secure communication. OpenSSL is considered an industry standard for secure and well-tested cryptographic implementations of many cryptographic algorithms defined in various standards and RFCs. The TOE does not use any proprietary implementation

of cryptographic standards. The following SFRs of the TOE are realized by using OpenSSL:

- FCS_CKM.1/rsa
- FCS_CKM.1/random
- FCS_CKM.1/password_stretching
- FCS_CKM.1/derivation
- FCS_COP.1/hash
- FCS_COP.1/keyed_hash
- FCS_COP.1/key_derivation
- FCS_COP.1/enc_symmetric_noauth
- FCS_COP.1/enc_symmetric_auth
- FCS_COP.1/enc_asymmetric
- FDP_DAU.1
- FIA_SOS.2

The TOE requests and makes use of X.509 certificates signed by the cloud servers or public certificate authorities for certain purposes. The security model described in this document treats these as public keys and does not assume that the certificates issued by the cloud servers can be trusted. While the TOE checks the signatures of these certificates (which may result in failed operations if the certificates are invalid or have expired), validating these certificates and the cryptographic primitives used to do so are neither included in the SFRs, nor are necessary to ensure that the security requirements are met.

7.2 File management

TOE users can upload files to their encrypted containers with the `upload` command. When a file is uploaded, unique keys are generated by:

- Generating a new random key, that is then stored alongside the file as metadata, encrypted with another key in the container.
- Deriving a new key from another key in the container, storing the parameters of the derivation alongside the encrypted file as metadata.

The keys are then used to encrypt the contents and name of the file with either:

- An authenticated encryption algorithm to protect the integrity and the confidentiality of the file, with the tag stored alongside the encrypted file as metadata.
- A non-authenticated encryption algorithm to protect the confidentiality, and a keyed hash to protect the integrity of the encrypted file, with the digest stored alongside the encrypted file as metadata.

The encrypted file alongside all its metadata is then uploaded to cloud servers for storage and later retrieval. There is no strict one-to-one mapping between files uploaded by the user and objects stored on the server – a single file uploaded by the user may be represented by multiple objects on the server.

When TOE users download files using the `download` command, the process is reversed – the unique keys that belong to the file are recovered by:

- Decrypting the encrypted key stored alongside the file as metadata using another known key in the container.
- Deriving the key again from another known key in the container, using the parameters of the derivation stored alongside the encrypted file as metadata.

The recovered keys are then used with the same primitive that was used for encryption and integrity protection to decrypt and verify the integrity of the file. The decrypted contents of the file are saved to the local storage of the user.

Directories are treated as special files by the TOE that store metadata about other files, and all statements that relate to files are also applicable to directories unless otherwise specified. Each container has exactly one root directory. All files uploaded must belong to exactly one directory, except the root directory (and the container key file, see next section). No two files in a directory may have the same name – if a file is uploaded with the same name as an existing file in a directory, that results in an error instead of the TOE uploading the file. Whenever a file is downloaded or uploaded, its parent directory may also be downloaded, modified, and then uploaded again to reflect the changes in the directory. Upload and download operations are performed on directories recursively – directories are uploaded and downloaded as normal files would be during these operations.

TOE users can list files in a directory using the `get-directory-contents` command. When this command is issued, the directory is non-recursively downloaded to memory, the decrypted contents are parsed and are used to display the list of files and directories in the directory to the user.

Empty directories may be created with the `create-directory` command. The TOE uploads such directories as if the user initiated the upload of an empty directory with the specified name.

Commands have optional and required path parameters to identify files to download or upload. Internal paths that refer to files inside a container are relative to the root directory of the container. These paths consist of the names of the parent directories and the name of the file separated by a path separator.

7.3 Container management

Containers TOE users have access to can be listed using the `get-containers` command, asking the server to return the names of all such containers. Container names are allocated by the cloud servers at the TOE's request. Containers may be shared between TOE users. Each container has a key file that stores metadata and encrypted container keys, which can be decrypted by a container key that is itself encrypted with the asymmetric user keys of each user who has access to the container. When required, the TOE downloads and decrypts these key files to access other keys required to decrypt data in the container in the background.

TOE users can create encrypted containers using the `create-container` command. After the cloud servers have allocated a name for the container, the key file is assembled and uploaded alongside an empty root directory. At this point, the key file's container key is only encrypted with the key of the user who created the container, making them the only user able to access and modify data in the container.

TOE users can share existing containers with other users. To do so, the user first must contact the invitee to obtain the identifier and fingerprint of their asymmetric user key. Invitees can use the `get-own-fingerprints` command on their own TOE to obtain these values. The invitation can be issued using the `invite-user-to-container` command, specifying the e-mail address, the key identifier, and the fingerprint. The TOE will first query the servers for the asymmetric user key of the invitee and check if the fingerprint supplied matches the fingerprint of the key retrieved from the server. Once validated, the asymmetric user key is used to encrypt the container key, then this encrypted container key is added to the key file, which is uploaded to the cloud servers. The out of band fingerprint check prevents third parties from spoofing the invitee's public key and fooling the user into leaking the container keys by encrypting them with a public key under the attacker's control.

TOE users can list containers they were invited to using the `get-container-invitations` command. Users can ask the user who invited them to a container to use the `get-container-fingerprint` command to query the latest fingerprint of that container, that can then be used alongside the container's name with the `acknowledge-container-invitation` command to accept the invitation. The out of band fingerprint check prevents third parties from spoofing the invitation and fooling the user into uploading data to a container the attacker has prepared with known keys.

TOE users can revoke access from other users to existing containers they have access to. To do so, the user first must query the fingerprint identifying

the user they want to remove from the container using the `get-user-fingerprints-in-container` command. The command will return fingerprints which do not necessarily match the fingerprints of the user key of the users when they were invited, as their user keys may have changed since then. The command will also try to display the e-mail addresses that belong to the users associated with these fingerprints, indicating if their validity could be cryptographically verified. The user may also verify these fingerprints by asking the users through another channel to match them with the output of the `get-own-fingerprints` command. Then, the user can issue the `kick-user-from-container` command, which will remove the container key in the key file encrypted with the key of the user being removed, then rotate the key, encrypting it again for all remaining users in the container.

Lazy re-encryption is utilized by the TOE for all data stored in the container. If the container key encrypted with the users' asymmetric keys changes, either because a user was removed from the container or one of the asymmetric keys changed, existing data remains encrypted with the old key. All new data and metadata uploaded to the container from that point on will be encrypted with the new key, or with a key that is encrypted by or derived from the new key.

When a user creates a container, or accepts an invitation to a container, the fingerprint of the container is encrypted using an authenticated encryption algorithm with a symmetric user key, stored with the user's profile, and later decrypted and used to check the integrity of the container key file. This check also ensures that only containers the user has acknowledged an invite for (or has created) are treated as belonging to the user. Container fingerprints generated at a certain time can validate the integrity of the current or any future key file associated with the container. The encrypted fingerprint is periodically updated with the latest fingerprint (after the integrity of the key file has been validated) to speed up this validation.

7.4 User management

The cloud servers used by the TOE require users to establish an account before any other functionality of the TOE can be used. TOE users can initiate the creation of a new user account with the `register` command. To create a new account, the user needs to provide an e-mail address, a first and a last name, as well as a password. The supplied password is checked to ensure that it meets the minimum complexity requirements. If requirements are met, the password goes through a password stretching algorithm to protect TOE users against brute force attacks.

The stretched password is used to encrypt a newly generated user key, which is used by the TOE with an authenticated encryption algorithm to protect the confidentiality and integrity of the user's profile. The user's profile is a collection of objects stored on the cloud servers. When stored on cloud servers, all other user keys are either stored as profile objects, encrypted with other user keys, or derived from other user keys. Objects in the user's profile are uploaded and downloaded in the background by the TOE whenever other operations require their contents.

The TOE then generates an asymmetric user key that can be used by other TOE users to invite the user to containers. The private part of this key is added to the user's profile. This user key is rotated in the background by the TOE periodically and after certain operations. When this key is rotated, the key files of all containers the user has access to are updated to reflect this change, triggering lazy re-encryption as described in the previous section.

The stretched password is also used to generate a token that can be used by the server to check the identity of the user. The user's name, e-mail address, the token, the public part of the asymmetric key that can be used by other TOE users to invite the user to containers, and the initial contents of the user profile are all sent to the cloud servers, which will send an e-mail to the user's address with a link the user is expected to open to validate that the user has control of the e-mail address that was supplied. The account cannot be used until this validation is completed.

TOE users can use the `login` command to sign into their accounts and start using the service. Users must provide their e-mail address and password to log in. The password is stretched, then the stretched password is used to recover the user's token and the key to decrypt the user key that was used to encrypt the user's profile. The user's e-mail address and the token are used with the cloud servers to establish a session that will be used to authenticate subsequent requests (the session-based authentication is used as an additional security measure, like TLS, and is not required to ensure that the security requirements described in this document are met). A new asymmetric user key is generated and is used to encrypt the user key to the user's profile. The public part of this key is sent to the cloud servers and tied to the session that has been established. The password, the token and the key derived from the stretched password that was used to decrypt the user profile is no longer used after this point.

All operations with the password during login and registration are performed locally. The raw password, the stretched password and the keys derived from the stretched password never leave the TOE.

TOE users can issue the `logout` command to terminate their session with the cloud servers.

7.5 Secure communication

The TOE uses state-of-the-art cryptographic solutions defined in SFRs mapped to this functionality to secure the confidentiality and integrity of transmitted data between TOE clients. Data protected with these cryptographic solutions are end-to-end encrypted between the TOE clients, therefore their confidentiality and integrity are protected against the cloud servers too.

In scope of the TOE and its documentation, export shall be understood as any data leaving the TOE towards the network and/or the Internet, and import shall be understood as any data entering the TOE from the network and/or the Internet.

8 Glossary of Terms

The document uses terms and definitions defined in the official Common Criteria documentation.

9 Acronyms

Acronym	Meaning
RFC	A Request for Comments (RFC) is a publication in a series, from the principal technical development and standards-setting bodies for the Internet, most prominently the Internet Engineering Task Force (IETF).

10 Bibliography

[CC_P1] Common Criteria, Part 1: Common Criteria for Information Technology Security Evaluation, Part 1: Introduction and General Model, Version 3.1, Revision 5, April 2017, CCMB-2017-04-001

[CC_P2] Common Criteria, Part 2: Common Criteria for Information Technology Security Evaluation Part 2: Security Functional Components, Version 3.1, Revision 5, April 2017, CCMB-2017-04-002

[CC_P3] Common Criteria, Part 3: Common Criteria for Information Technology Security Evaluation, Part 3: Security Assurance Components, Version 3.1, Revision 5, April 2017, CCMB-2017-04-003

[CEM] Common Methodology for Information Technology Security Evaluation, Evaluation Methodology, Version 3.1, Revision 5, April 2017, CCMB-2017-04-004

[SP800-38A] SP 800-38A – Recommendation for Block Cipher Modes of Operation: Methods and Techniques, 2001

https://doi.org/10.6028/NIST.SP.800-38A

[SP800-38D] SP 800-38D – Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC, 2007-11

https://doi.org/10.6028/NIST.SP.800-38D

[SP800-56Br2] SP 800-56B – Recommendation for Pair-Wise Key-Establishment Schemes Using Integer Factorization Cryptography, rev. 2

https://doi.org/10.6028/NIST.SP.800-56Br2

[SP800-90Ar1] SP 800-90A – Recommendation for Random Number Generation Using Deterministic Random Bit Generators, rev. 1

https://doi.org/10.6028/NIST.SP.800-90Ar1

[SP800-133r2] SP 800-133 – Recommendation for Cryptographic Key Generation, rev. 2

https://doi.org/10.6028/NIST.SP.800-133r2

[FIPS180-4] FIPS 180-4 – Secure Hash Standard (SHS), 2015-08

https://doi.org/10.6028/NIST.FIPS.180-4

[FIPS198-1] FIPS 198-1 – The Keyed-Hash Message Authentication Code (HMAC), 2008-07

https://doi.org/10.6028/NIST.FIPS.198-1

[RFC7914] RFC 7914 – The scrypt Password-Based Key Derivation Function, 2016-08

https://datatracker.ietf.org/doc/html/rfc7914

[RFC8018] RFC 8018 – PKCS #5: Password-Based Cryptography Specification Version 2.1, 2017-01

https://datatracker.ietf.org/doc/html/rfc8018

[GDPR] General Data Protection Regulation

Requirement source: Section 2: Article 32 & Article 34

https://eur-lex.europa.eu/legal-content/EN/TXT/ELI/?eliuri=eli:reg:2016:679:oj

[HIPAA] Health Insurance Portability and Accountability Act of 1996

Requirement source: Policy 3.2 & 45 CFR § 164.312

https://www.cdc.gov/phlp/publications/topic/hipaa.html

[TISAX] TISAX Participant Handbook

https://enx.com/tphen.pdf

[BSI_C5] Cloud Computing Compliance Criteria Catalogue – C5:2020

Requirement source: Confidentiality of Authentication Information (PSS-07), Secure key management (CRY-04), and Access to cloud customer data (IDM-07)

https://www.bsi.bund.de/SharedDocs/Downloads/EN/BSI/CloudComputing/ComplianceControlsCatalogue/2020/C5_2020.pdf

[CJIS] Criminal Justice Information Services (CJIS) Security Policy

Requirement source: 5.5 Access control and 5.10.1.2 Encryption

https://www.fbi.gov/file-repository/cjis_security_policy_v5-9_20200601.pdf/view

[CMMC] Cybersecurity Maturity Model Certification 2.0

Requirement source: AU.L2-3.3.8, MP.L2-3.8.6, MP.L2-3.8.9 and SC.L2-3.13.8

https://www.acq.osd.mil/cmmc/

[FEDRAMP] FedRAMP Security Controls Baseline

Requirement source: MP-5(4), SC-8(1) and SC-28(1)

https://www.fedramp.gov/assets/resources/documents/FedRAMP_Security_Controls_Baseline.xlsx

[FINRA] Report on Selected Cybersecurity Practices – 2018

Requirement source: Technical controls and Data loss prevention sections

https://www.finra.org/sites/default/files/Cybersecurity_Report_2018.pdf

[ISO27001] ISO/IEC 27001:2022 Information security management systems

Requirement source: 7.5.3

https://www.iso.org/standard/27001

[ITAR] 22 CFR § 120.54 - Activities that are not exports, reexports, retransfers, or temporary imports

https://www.law.cornell.edu/cfr/text/22/120.54

[SP800-53] NIST Special Publication 800-53 Revision 5 - Security and Privacy Controls for Information Systems and Organizations

Requirement source: SC-12, SC-13, AU-9(3), MA-4(6), SC-8(1), SC-8(2), SC-8(3) & SC-8(4)

https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-53r5.pdf

[SOC2] TSP Section 100 - 2017 Trust Services Criteria for Security, Availability, Processing Integrity, Confidentiality, and Privacy

https://us.aicpa.org/content/dam/aicpa/interestareas/frc/assuranceadvisoryservices/downloadabledocuments/trust-services-criteria.pdf

[DTL] Criteria Catalogue for the Digital Trust Label

Requirement source: No. 1 & 2

https://digitaltrust-label.swiss/wp-content/uploads/2022/02/DTL-Criteria-Catalogue.pdf