PRIM'X TECHNOLOGIES

Version 6.1

Security Target

Common criteria EAL3+

Document v1 r9

Zed!
FOR SECURE DATA EXCHANGE

Table of contents

List of figures

List of tables

1. Introduction

1.1. ST identification

Security Target :	Zed! version 6.1 build 2120 Security target CC EAL3+
ST version :	PX159534 v1r9 – March 2016
Target of Evaluation (TOE) :	- Zed! v6.1 Standard Edition - Zed! v6.1 Limited Edition for PC platforms running Microsoft Windows Seven and Windows 10 (64 bits)
EAL level :	EAL3 augmented with ALC_FLR.3 and AVA_VAN.3 components associated with analysis of the cryptographic implementation described in [QUALIF_STD].
Conformance with a PP :	None.
CC reference :	Common Criteria version 3.1 Revision 4, Parts 1 to 3 – September 2012

1.2. TOE overview

Zed! is a software security product designed for manufacturing encrypted (and compressed) file containers either for archiving or for exchange with correspondents as email attachments or on various portable devices, such as USB sticks.

Zed! will be evaluated for a PC platform with Microsoft Windows Seven and Windows 10 operating systems (64 bits).

1.3. Common Criteria conformance

This security target is compliant with Common criteria requirements version 3.1, September 2012:

[CC1] Common Criteria for Information Technology Security Evaluation, Part 1: Introduction and general model. Version 3.1, Revision 4, September 2012. CCMB-2012-09-001.

[CC2] Common Criteria for Information Technology Security Evaluation, Part 2: Security functional requirements. Version 3.1, Revision 4, September 2012. CCMB-2012-09-002.

[CC3] Common Criteria for Information Technology Security Evaluation, Part 3: Security Assurance Requirements. Version 3.1, Revision 4, September 2012. CCMB-2012-09-003.

[CEM] Common Methodology for Information Technology Security Evaluation, Evaluation Methodology. Version 3.1, Revision 4, September 2012. CCMB-2012-09-004.

All functional components described in this security target are derived from the part 2 "strict" of the common criteria version 3.1 revision 4 of September 2012. The selected assurance level "EAL3 augmented" is consistent with part 3 "strict" of the common criteria version 3.1 revision 4 of September 2012. The assurance level is a level EAL3 augmented with ALC_FLR.3 and AVA_VAN.3 components.

All interpretations of the common criteria published on the date the evaluation is starting will be considered.

1.4. ANSSI referential conformance

This security target is compliant with following ANSSI referential:

[QUALIF_STD] Processus de qualification d'un produit de sécurité – niveau standard – version 1.2, DCSSI.

[CRYPTO_STD] RGS version 2.0 – Annexe B1. Mécanismes cryptographiques : Règles et recommandations concernant le choix et le dimensionnement des mécanismes cryptographiques - Version 2.03 du 21 février 2014, ANSSI.

[CLES_STD] RGS version 2.0 – Annexe B2. Gestion des clés cryptographiques : Règles et recommandations concernant la gestion des clés utilisées dans les mécanismes cryptographiques - version 2.0 du 8 juin 2012, ANSSI

[AUTH_STD] RGS version 1.0 – Annexe B3. Authentification : Règles et recommandations concernant les mécanismes d'authentification - Version 1.0 du 13 janvier 2010, ANSSI.

2. TOE description

2.1. TOE presentation

2.1.1. General description

Nowadays emails are widely used by companies to communicate internally or with their partners and providers. This communication on a daily basis induces systematic exchange of sensitive documents. But, most of the exchanged documents are simply attached to emails, thus guaranteeing no confidentiality for the transmitted data.

Zed! is a **security product** for computers running Windows, Linux and Mac operating systems (32 and 64 bits). The Zed! product (available independently or integrated in ZoneCentral) is designed for manufacturing **encrypted (and compressed) file containers** either for archiving or for exchanging with correspondents as email attachments or on various portable devices, such as USB sticks. The product also incorporates a mechanism for checking the integrity of the files stored in the containers.

The use of encrypted containers is highly intuitive and very similar to the use of "zipped folders" under Windows. The user can copy some files in the container, rename them, delete them, extract them, etc. Zed! has no file size limit and does not modify the tree view of files or folders that it copies.

The container can be used as an encrypted storage of files, without modifying their characteristics (location, name, date and size), in the most possible transparent way for users. Files encryption/decryption is performed when these files are read/copied in the container and *'on the fly'* (without special handling by the user).

Once the user has created a container, he/she can add accesses for his correspondents. An access corresponds to an **access key** (cryptographic key) that a user owns. This key can either be a password in this case the user does not hold the access key itself but the password allowing Zed! to calculate it) or an RSA key stored in a key holder as a key file (RSA certificate found in a certificate file or in LDAP directories), a smart card, or a Microsoft Windows CSP orCNG container (the key holder can itself be protected by a PIN code). An access key is used to find (by decrypting) the encryption information pertaining to container. Zed! also features a very practical function: the 'Password Wallet', which is used to store and reuse the different passwords of correspondents. And finally, Zed! offers the possibility to hide the file names in the container: the container seems empty until a valid access key has been entered.

Zed! is available in a variety of packages:

- Zed! Standard Edition, which contains the complete product
- Zed! Limited Edition, which is free of charge and free for distribution and usage, and can be used for reading the content of containers and extracting the files (providing that the correspondent enters a valid access key). The correspondent can also modify the container (add or suppress some files) before sending it back to the sender. However, the limited

edition does not allow the creation of new containers or the access modification (manage by the original creator of the container). The limited edition comes in the form of a simple executable file (zedle.exe), easily portable, eliminating the need to install the program

- Finally, Zed! is also incorporated in the various products of the ZoneCentral range.

2.1.2. Zed! technology

The internal format of the container is designed to contain files of any size, independently of each other, and it handles the naming of these files. The container can be considered as a "virtual folder": its content in terms of files is not confidential, files content is. The container contains a control file that manages the accesses and is hidden.

2.1.3. Container and accesses

Each encrypted container is defined by some encryption characteristics (including file encryption keys, algorithms, etc.) and a user access list. The definition of the accesses is free, but the product has administration functions and mechanisms to impose certain accesses or certain access types.

To use a container, a user must have an access key. In the case of Zed! Standard Edition, the user will be given this key by the Security Officer (called TOE Administrator in this document). It may be a RSA key hosted in a key holder like a key file, or a smart card, a Microsoft CSP or CNG container (the key-holder incorporating most of the time its own authentication device with a confidential code). Most of the time, the password is chosen by the user depending on the security policy of the company.

Once the user is authenticated, the access key provided remains valid as long as it has not been explicitly closed by the user (lock in the standard edition, closure of the container in the limited edition, session logout or shutdown for example).

When the encrypted container was created, each file protected by the container was encrypted using keys, and these keys were themselves encrypted with the user access keys. Obviously, access keys themselves are not stored in the container.

Zed! offers different state-of-the-art security algorithms and mechanisms. It provides two systems for managing access keys. These systems can be used at the same time on the same containers: A so-called 'symmetrical' system based on passwords and keys derived from passwords (ref.: PKCS#5), and a so-called 'asymmetrical' system that uses RSA keys (ref..: PKCS#1 v1.5) embedded in key files (ref.: PKCS#12) or physical objects (ref.: PKCS#11 and/or CSP/CNG).

2.2. Administration services and roles

2.2.1. Role definition

Apart from the security officer of the organization which sets the security policy, there are 4 roles implementing (directly or indirectly) the functionality of the TOE:

- A role operating in the TOE environment only (this role does not concern Zed! Limited Edition[1]): the security administrator of the Windows environment (Windows administrator) in charge of defining rules for use and policies, i.e. the setting of the product: this "high-level" operation is performed under the supervision of the security officer who studied the different parameters and decided values that must be assigned to get the desired behavior of the product. Policies are signed by the security administrator and verified by Zed! prior to their application: the policy signature mechanism guarantees that only policies validated by the administrator are applied on the workstations. While a domain administrator is authorized to modify the domain's policies, he/she will be unable to change the Zed! configuration: if he/she modifies any policies, the signature will become invalid and, as a result, new policies will be refused on the workstations. Once applied, rules will change only in very exceptional cases. It should be noted that this role may be declined in several hierarchical roles corresponding to the different levels of the Windows domains. In this case the upper levels administrators must prohibit modification of the TOE policies by the sublevels administrators (domains, domain controllers, workstations).
- The **security administrator** role (this role does not concern Zed! Limited Edition[1]) in charge of TOE installation, policy signature, recovery operations (including SOS procedure), access key and possibly password distribution to users. The security administrator can also manage the accesses.
- The **access administrator** (this role does not concern Zed! Limited Edition[1]) in charge of the access management, that is the creation of the first access when the container is initialized (namely his/her own access) and the addition or suppression of other accesses.
- A **user** role that uses the TOE according to the configuration imposed by the administrators (this role is generally devoted to the correspondents when the container are exchanged).

It should be noted that, apart from the policies definition usually devolving to a security officer, the other operations can be performed by different actors depending on trust and organization.

[1] Since the limited edition cannot perform management operations. However the containers sent to the owners of the limited edition have been configured by these roles.

2.2.2. Administration and operation services

There is no administration operation on a container apart from the access management the policy signature. A container 'lives' as long as it is not deleted.

Possible operations are (standard edition only):

- Displaying or modify the policies, signing the policies. Note that by default policies are not signed ; the prior operation of policies signature and their integration in the installation package are required to benefit from the signing function.
- The container initialization when the container is accessed for the first time (corresponds to the creation of the first access).
- Adding, modifying or deleting a user access (implying that the access key is provided).
- Modifying the access role
- Recovery by the security administrator
- User SOS procedure by the security administrator

Note: The visualization of the container accesses does not require the administrator role and is available in the limited edition.

Possible operations are:

- The creation of a new container (by the user with the standard edition): this operation does not require any access key. This operation is not implemented in Zed! Limited Edition.
- Renaming or deleting a container (in the Windows environment).
- Adding files to the container (drag & drop, insert, copy and paste, etc.): files are encrypted when they are put in the container, which requires to have provided the container access key.
- File visualization or files extraction in a container: any "output" of file requires the container access key so that the files can be decrypted in the container.
- The deletion of files in container.
- Creating and deleting files in the container.

User interface of encrypted containers is similar to compressed files (.zip) in Windows (drag & drop, insert, copy and paste …).

2.2.3. Use case

Whilst there are various deployment scenarios, the underlying principle remains the same for users and applications.

The Security Officer defines the product's ***policies*** and signs them with his/her private key. This involves a predefined configuration (policy) that can be mastered (customization of the installation) or remotely managed (distributed, updated), either using administration commands provided by the product, or the logistics

integrated in office automation networks (e.g., domain controllers). These policies are usually established on a 'high level' within the company by the Security Officer. The policies define, for example, which encryption algorithm to use, the behavior the product must adopt in certain cases, the PKCS#11 key holders supported etc.

The software, mastered or not, is then ***installed*** on a workstation either manually, or via download applications available on the market.

Additionally, it is up to the TOE administrator to **define (i.e. provide) the user access keys** (e.g., from a PKI). Zed! supports several key management scenarios, but does not provide the corresponding infrastructure. If a PKI is installed, Zed! knows how to use its elements (RSA keys, key holders, certificates). If a PKI is only partly installed, or if there is no PKI, Zed! also knows how to use password access.

When a user creates a new container with the standard edition, he/she becomes the access administrator, defining the accesses and the roles of its correspondents (by default the role is simply "user")

Then, only users with valid access keys are allowed to read or copy files in the container. At the first attempt to access an encrypted file in the container, Zed! asks for the user access key to decrypt the file (in fact, the schema is more complex, and this access key is used to decrypt some intermediate keys which themselves encrypt files). If the user can provide it, then the file can be decrypted (or encrypted, if it is a creation or a write operation). Otherwise, the access is denied with the usual error code "unauthorized access". The user authentication is valid for this container until the session is locked or put in sleep mode (standard edition only), the session is log out or the workstation is shutdown. As for the zip archives, to "open" (read) a file of a container, it must first be extracted. The "open" operation automates the extraction (in the user's temp folder) and activates it. When the container is closed, the file is encrypted.

2.3. TOE scope and architecture

2.3.1. Edition standard components

Figure 1 shows the architecture of the product: the scope of the TOE is bounded by dashed lines. The main components are the following:

- « **Interface Explorer** » implements Windows Shell interfaces to manage menus and graphics view.
- « **ZDU** » is a user daemon, instantiated for each Windows user session, referencing the user keys entered via password, PKCS#11 interface, CSP or CNG.
- The « **ZEP** » service controls policies signatures.
- « **Zed Engine** », coordinates all the treatments;
- The « **crypto driver** » is the Zed! cryptographic centre: it manages container keys and performs associated calculations. Keys are never stored out of its "enclosure", except when the product is configured to use key holders (as

PKCS # 11 extensions for smart cards or CSPs/CNGs). This implementation of the cryptography in kernel-mode strengthens the overall protection level because the kernel is a location hardly accessible to hacking softwares.

The « **keyboard driver** » is a keyboard filter: it intercepts at very low level passwords and confidential codes entered by the user. In this way their values remain confined at the lowest possible level in the system. These secrets are then used by the cryptographic driver or given to external engines (CSP/PKCS # 11). This only concerns passwords managed by Zed !, i.e. those which guaranty access to the encrypted files. This implementation also strengthens the protection of sensitive data: they are not sent back to the application level of the system in a location which is the regular and favorite source of hacking softwares.

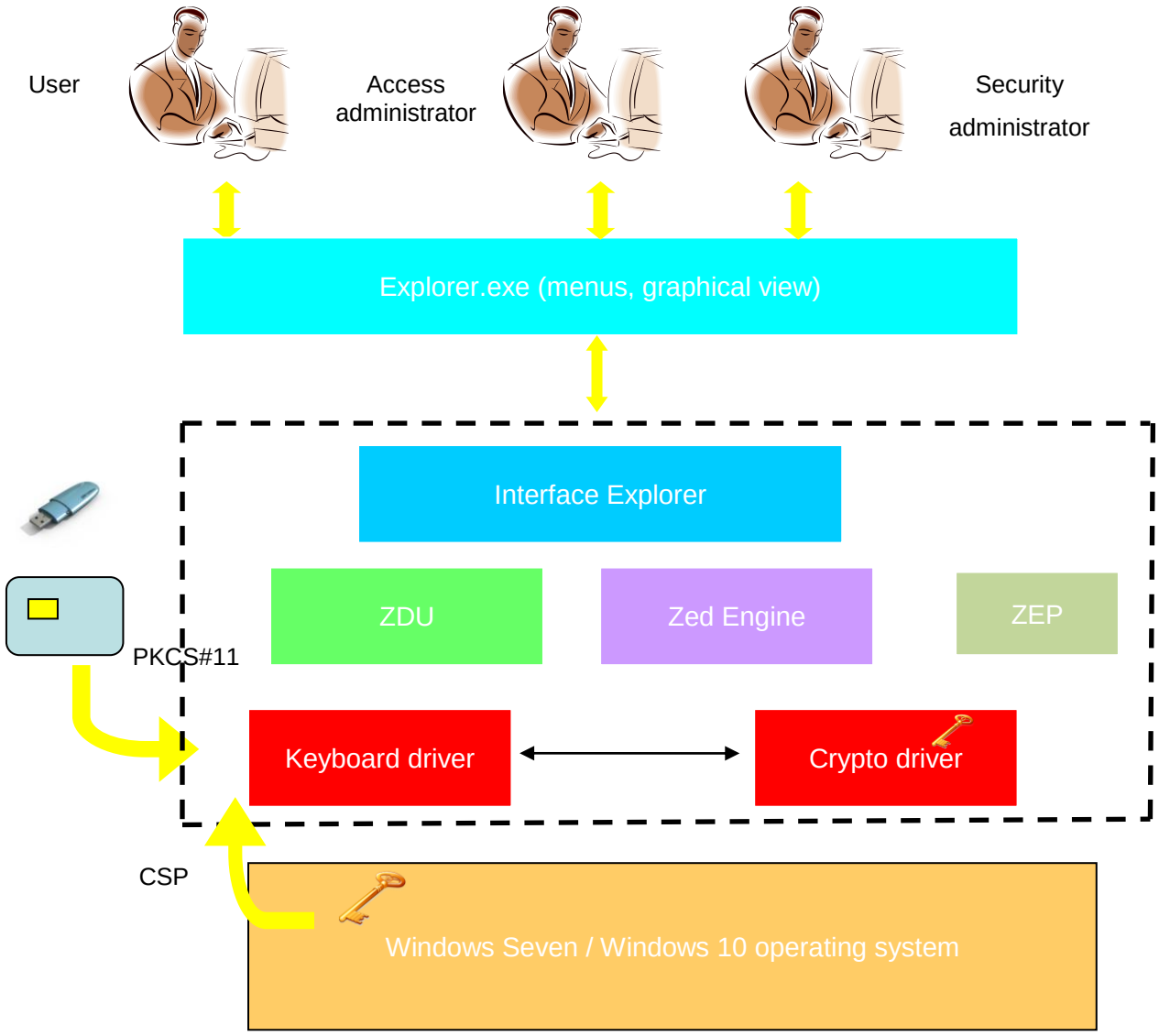

Figure 1– Standard edition architecture

2.3.2. Edition standard components

Figure 2 shows the architecture of the product: the scope of the TOE is bounded by dashed lines. The main components are the following:

- « **BROWSER** » emulates Windows Explorer.
- « **Interface Explorer** » manages menus and graphics view.
- « **ZCC** » is the cryptographic centre of Zed! : it manages keys and performs associated cryptographic operations. Keys are never stored outside the centre, except when the product is configured to use key holder (as PKCS#11 extensions for smart cards and CSPs).
- « **ZCCA** » references the user keys entered via a password entry, the CSP or the PKCS#11 interface.
- « **Zed Engine** » coordinates some treatments;

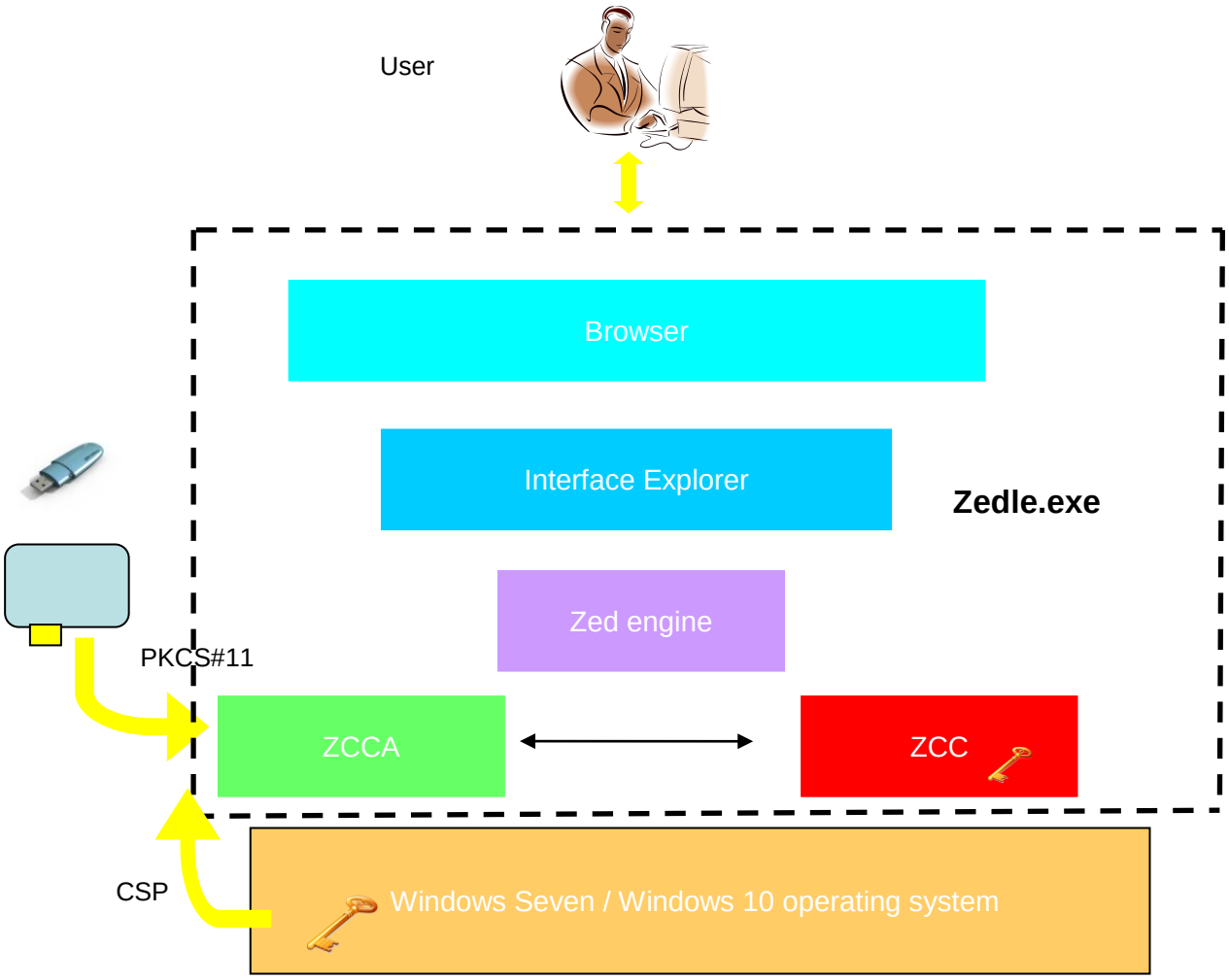

Figure 2 – Limited edition architecture

Note that the limited edition does not perform any operation administration operation.

2.3.3. TOE scope

2.3.3.1 Logical scope

The TOE includes Zed! Standard Edition and Zed! Limited Edition. Zed! integrated in ZoneCentral is not in the evaluation scope.

Only the build 2120 configured with the following security policies is compliant:

- The P730 policy (password acceptance threshold) must be set to 100% and the P732 policy (password length) to 12.
- The P381 policy (encryption Mechanism for encrypted containers) must be set to « CTS » (default value).
- The P399 policy (Format version of encrypted containers and messages) must be set to « Version 2 ».
- The P383 policy (RSA Encryption Scheme) must be set to « PKCS#1 v2.2 with SHA-256 ».
- The P382 policy (enable use of AES-NI instruction set) must be set to « No » (default value).
- The P233 policy (hiding file and folder names of encrypted containers) must be set to « Always hide ».
- The P292 policy (Hash algorithm used) must be set to « SHA-256 » (default value)

The evaluation scope is made up of all the components of the software apart from the following features:

- The GPOSign.exe tool used by the security administrator to sign policies and the generation of the key to perform this signature. However the signature verification of the Zed! policies is part of the TOE scope.
- The use of SSO (Single Sign On) mode that allows to automatically open the encrypted containers when the Windows session is open (but shifts the security level to Windows or to the third-party SSO component).

2.3.3.2 Physical scope

Zed! will be evaluated, as a product, on a PC platform under the following Microsoft operating systems: Windows Seven and Windows 10 (64 bits).

Use with different access keys will be evaluated (password and RSA key). In particular, PKCS#11 dialog between the TOE and the user key holders, PKCS#12 dialog between the TOE and the key files will be also evaluated.

The following elements are not evaluated;

- Windows operating systems;
- Key holders such as USB tokens, key files or CSP/CNG containers.

Zed! software uses user keys (the "access keys") provided by the environment (RSA keys in key holders or passwords provided by the TOE administrator).

2.4. Test platform used for TOE evaluation

To perform the evaluation of Zed!, the following minimum platforms will be set by the evaluator (PC stands for virtual machine). The physical type of key holder (smart card or USB key) being transparent to Zed! (only the PKCS#11 dialogue is important), the evaluator tests will be performed with a single type of key holder.

Security policies will be activated in accordance with the logical scope defined above.

Policies signature feature requires installation of the TOE with a specially prepared installation package and careful reference to the documentation for the function.

Platform 1 running Windows Seven 64 bits with standard edition only (build 2120) and authentication using USB token:

- A domain controller (Windows 2008R2)
- One or two PC running Windows Seven 64 bits with standard edition (build 2120): a single PC can be used to perform the operations.

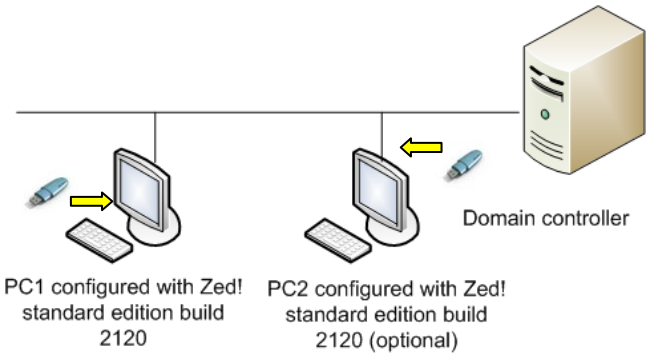

Figure 3 – Test platform with Windows Seven

Platform 2 running Windows 10 64 bits with standard edition and limited edition (build 2120) and authentication using passwords:

- A domain controller (Windows 2008R2)
- A PC running Windows 10 64 bits with standard edition (build 2120) : container creation and access management, use of containers
- A PC running Windows 10 64 bits with limited edition (build 2120) : use of containers

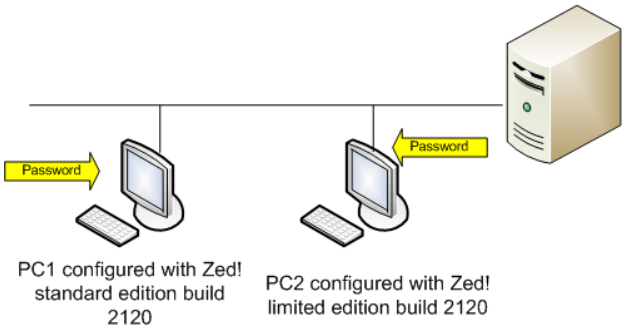

Figure 4 – Test platform with Windows 10

3. Security problem definition

3.1. Assets

3.1.1. User assets

3.1.1.1 Access keys: D.USER_AUTH

To open any encrypted containers (reading a file, copying a file, access management) Zed! need user access keys. According to circumstances, it may manipulate the user password, or the access key itself, or its confidential protection code.

- Password access: Zed! manages the password entry, its transformation (derivation) into access key and then the decryption of the encryption key and the decryption of files encrypted by this access key. The minimum password strength is configurable in the security policies.
- Access by RSA key hosted in a key file using the PKCS#12 mechanism: Zed! manages the confidential code entry of the key file, reads and decrypts the key file with this confidential code, gets the RSA access key and performs the decryption of the encryption key and the decryption of files encrypted by this access key.
- Access by RSA key in a logical token accessed through an external PKCS#11 component (this component can handle a memory card, a USB token or any other hardware or software device): Zed! manages the confidential code entry of the logical token, submit it to the external component to unlock it. Zed! also provides the external component with the encryption key of files encrypted by the public key. The component decrypts the encryption key using its private key and then transmits it to Zed! which can then perform the decryption of files.
- Access by RSA key hosted in a logical token accessed through an external component CSP or CNG (this component can handle a memory card, a USB token or any other hardware or software device): Zed! does not manage the confidential code of the logical token entry, it is the external component which spontaneously does it with its own means, and Zed! does not access the RSA key and does not perform the unwrapping of the files encryption key. It is performed by the external component which then transmits the unwrapped key to Zed!.

Based on these cases, therefore, Zed! manipulates the following sensitive assets: a password (or a PIN code), and a cryptographic access key. In cases 1 and 2, it manipulates the two elements, in cases 3 it manipulates the first one, in case 4 no element is manipulated by Zed!.

It should be noted that Zed! does not generate user access keys: as regards RSA keys, regardless of the key holder that hosts them and the module that deals with them, they are always generated by an external tool (generally a PKI), as well as the potential key holder and the associated confidential code. As far as passwords are concerned, they are chosen by the security administrator or the first user (access administrator). The user and its environment (rules and internal procedures, established by the security officer) are responsible for the quality of these keys, the protection of the key holder and their proper use.

3.1.1.2 Signature key pair : D.ID_ADMIN

Security policies are signed by the security administrator and verified by Zed! before their application. The signature key pair (especially the private key) of the administrator is part of the sensitive assets of this particular user.

3.1.1.3 Encrypted files: D.USER_DATA

Zed! allows to encrypt files and folders in a container (possibility to hide the file names). Sensitive assets are therefore user files and folders, of all types, stored in encrypted containers.

Encrypted files in encrypted containers are sensitive user assets protected by the TOE (which should preserve their stored image in encrypted form without clear copy) as long as they remain in their encrypted container. An integrity check is performed when the user open a file in the container.

3.1.2. TOE assets

3.1.2.1 Symmetric keys used to encrypt the files: D.FILE_KEYS

The files of the container are encrypted by an encryption key generated when the container is initialized. A specific initialization vector is associated to each file. These assets are encrypted by the access keys and stored in the control file of the container.

3.1.2.2 Programs : D.PROGRAMS

The TOE operates through its ***programs*** (executables, dynamic libraries). The integrity protection of these programs is provided by the environment: it is necessary to be a Windows administrator to modify them. These programs are also signed (authenticode Windows system).

3.1.2.3 Configuration : D.CONFIGURATION

The TOE also operates through policies (this is the Windows term). The integrity protection of these policies is ensured:

- By the environment (i.e. the Windows policies system): only the highest level Windows administrator can modify them (if a Windows domain defines a value for a parameter, then a local administrator of the workstation will not be authorized to change it).

- By the product since policies are signed by the security administrator and verified by Zed! prior to be applied.

3.1.2.4 Control file: D.CONTROL_FILE

This file contains the container label, a unique identifier, some management information, and the access 'wrappings', i.e. the container encryption keys encrypted by user access keys authorized to the container. An integrity check is performed on this control file.

3.1.2.5 Catalogue file : D.CATALOGUE_FILE

This file contains the list of the application files in the container, with their position in the tree view, the original sizes, timestamps, etc. An integrity check is performed on this catalogue file.

3.1.3. Sensible assets summary

The following table summarizes the list of sensitive assets protected by Zed! and indicates the nature of the associated sensitivity.

The qualifiers 'high' and 'low' of the sensitivity refer to the protection level against the attack potential referred in the target (Chapter 3.3). A high sensitivity requires a protection level resistant to the attack corresponding to the targeted level (asset disclosure, undetected integrity alteration), low sensitivity indicates that the asset does not have to be protected taking into account the targeted level. For example policies disclosure brings little interesting information to a potential attacker (general product configuration) but any policy modification must be controlled or the product security will be degraded (addition of a recovery access for example).

Note: integrity protection is not the main goal of Zed!. The product goal is to manage the confidentiality of sensitive assets. However, Zed! includes a mechanism that checks the integrity of the files before they are opened by the user. Moreover, Zed! implements functions to detect alterations which would be harmful to its functioning, or that would induce some defects in its objective to enforce confidentiality.

Assets	Confidentiality	Integrity
User assets		
Elements of the access keys used by Zed!: password or possible PIN (D.USER_AUTH)	High	NA
Elements of the access keys used by Zed!: access keys itself if they are directly used by Zed! (D.USER_AUTH)	High	High
Signature key pair (D.ID_ADMIN)	High	High
User files and folders stored in encrypted containers (D.USER_DATA)	High	*Low (but any error must be notified)*
TOE assets		
Symmetric keys used to encrypt the files : D.FILE_KEYS	High	High
Container control file (D.CONTROL_FILE)	*Low*	High
Container catalogue file (D. CATALOGUE_FILE)	*Low*	High
Zed! configuration (D.CONFIGURATION)	*Low*	High
Zed! Programs (D.PROGRAMS)	*Low*	High

Table 1 : Assets to protect

3.2. Assumptions

For Zed!, named "the TOE" in the following paragraphs, the following assumptions on the environment will be taken into account for the evaluation:

A.NON_OBSERV The physical environment for the use of the TOE enables users to enter their passwords (or PIN numbers) without being directly observable and protecting them from other users or hackers to intercept their passwords or PIN numbers (wireless keyboard, etc.).

A. SECURE _PC The operational environment does not allow any attacker to access the hard disk when sensitive data are processed on the workstation by an authorized user. The user workstation must ensure effective protection against eavesdropping and unauthorized data transmission (correctly configured firewall, up-to-date antivirus software, anti-spyware, etc.).

A.TRUST_ADMIN	The security administrator and the access administrator are trustworthy. Windows administrators are trustworthy people responsible for the policy configuration (with secure values). All these people and the users are trained to the TOE usage. If the correspondents belong to entities managed by different Windows environment security administrators, the latter must conjointly guarantee the use of security policies in compliance with the requisite levels (in particular, password strength).
A. KEY_STORAGE	Users are responsible for the safekeeping in a secure location and for the non-disclosure of access keys that are sent to them by a TOE administrator. The TOE administrator is responsible for the safekeeping in a secure location and for the non-disclosure of the recovery access keys and his/her signature key.
A.CERTIFICATES	When access keys that possess an X509 certificate are supplied, the TOE administrator must verify that these certificates are in fact valid and suitable for TOE usage.
A.ENV_PROTECT_TOE	The technical environment of the TOE ensures the integrity of the TOE components. The TOE administration and update are carried out by traine and authorized people.
A.LOYAL_ENV	The runtime environment provides the TOE with exact date and time to ensure time stamp functions.
A.ENV_RNG	The TOE implements mechanisms to provide the random numbers necessary for the generation of the secrets.
A.EXT_CRYPTO	The access keys generated or stored outside the TOE must be compliant with [CRYPTO_STD] document for the standard robustness level.

3.3. Threats [to TOE sensible assets]

We are considering the threats on the sensitive TOE assets. Those relating to the user assets are covered by the organizational security policies (the product services) described below.

We can consider that there are three types of threatening agents:

- **Unauthorized user** that accesses the data of the legitimate user by theft or illegal access to the workstation. We hypothesize that some other means are used to protect sensitive data stored on the user workstation (e.g. disk encryption). However the threatening agent should not be able to open containers use to store sensitive data on the computer or access sensitive data that have been temporarily stored in the TOE operational environment when Zed! was used by the authorized user .
- **Eavesdropper**, who intercepts the data on the network (without physical access to the workstation). For example, he/she intercepts a mail with a container attached and attacks the container by installing or not the Zed! product (or Zed! Limited Edition) on his/her workstation.
- **A person causing or taking advantage of a malfunction**.

The considered attacker has an "enhanced-basic" attack potential as defined in the Common Criteria.

T.COMPONENT_MISUSE	**A person** intercepts a container, get the Zed! product (or Zed! Limited Edition) and manipulates, possibly at low level, internal components of the TOE, to bypass some security functions by **causing or taking advantage of a malfunction**. He can do this by reverse-engineering programs, or by developing programs calling internal functions of the TOE, or by modifying the internal configuration of the TOE or by using a debugger. Impacted asset is the TOE program (confidentiality and integrity) and the configuration (integrity). By these different means, the attacker must not succeed to "enter" a container in which he/she has not access.
T.POLICIES_SECU_INT	**A person taking advantage of a malfunction** signs the policies in place of the security administrator (domain policies areas or local policies if the attacker can access the workstation). For example, he/she can configure his/her own recovery access which will be added automatically when the user will create the next containers. The impacted asset is the configuration (integrity).

T.CONTROL_FILE_CONF	**An eavesdropper or an unauthorized user** retrieves the control file of the TOE in an attempt to find protected information. The impacted asset is therefore the control file of the container (confidentiality). For example, the attacker tries to find some protected information (e.g. encryption keys) from the encrypted files of the container and from the TOE control file or tries to decrypt (brute force) the information stored in the control file.
T.CONTROL_FILE_INT	**An eavesdropper or an unauthorized user** retrieves the control file of the TOE (stored in the container) and modifies it in an attempt to add its own access among authorized accesses (the attacker can position itself between two correspondents for example). The impacted asset is therefore the control file of the TOE (integrity). The attacker can thus intercept and read the files exchanged between legitimate correspondents or send the container to a correspondent (by usurping the identity of a legitimate user) so that the correspondent send him sensitive files.
T.CATALOGUE_FILE_INT	**An eavesdropper or an unauthorized user** retrieves the catalogue file of the TOE (stored in the container) and modifies it in an attempt to modify the container tree view. The impacted asset is therefore the catalogue file of the TOE (integrity). The attacker can thus intercept the exchanged containers and make disappear one or more files in the container without any detection by the recipient.

3.4. Security objectives for the operational environment

OSP.CONFIDENTIALITY	The TOE shall offer automatic and systematic confidentiality protection (encryption) of user sensitive files that are stored or send (email attachment).
OSP.INTEGRITY	The TOE shall provide automatic and systematic integrity check (hash) of user sensitive files in the containers.
OSP.ACCESS	The TOE shall enable users to provide an access key allowing the access to sensitive files of the container in which they wish to access. If they cannot provide a valid key for the container, access must be rejected.
OSP.RECOVERY	The TOE shall offer a service to recover sensitive files of users by using recovery access keys managed by the security administrator. These keys are systematically and automatically assigned during container initialization. The TOE must also provide a remote recovery (SOS procedure) if the user forgot his/her password or lost/broken his/her token. This SOS procedure is performed with a key systematically and automatically assigned during the creation of the user access list. This policy applies only to Zed! Standard Edition.
OSP.ADMIN_ACCESS	The TOE shall offer a service to manage accesses (Zed! Standard Edition only).
OSP.POLICIES_VERIF	The TOE shall provide a service without (special handling by the user) that performs the verification of the signature of security policies signed by the security administrator's private key (Zed! Standard Edition only). The application of any new policy is conditioned by the success of this verification.
OSP.CRYPTO	The ANSSI referential ([CRYPTO_STD], [CLES_STD] and [AUTH_STD]) defined for the 'standard' robustness level must be applied for the key management and for the cryptographic and authentication mechanisms used in the TOE.

4. Security objectives

4.1. Security objectives for the TOE

4.1.1. Access control

O.AUTH The TOE shall identify and authenticate users. To do this, the TOE shall allow access to a container only after presentation of a valid key for the container.

O.ROLES The TOE shall manage three user roles in an encrypted container: a 'normal user' role or simply 'user' role (use of files in the container after presentation of a valid access key), a 'access administrator' role in charge of initializing the container and adding the authorized accesses and a 'security administrator' role (installation, policies signature, recovery, access management).

4.1.2. Cryptography

O.ENCRYPTION The TOE shall encrypt and decrypt sensitive data with cryptographic keys. The TOE shall use different encryption keys to protect the different containers. The TOE shall generate encryption keys according to requirements for the standard robustness level of the cryptographic referential [CRYPTO_STD] and [CLES_STD] of ANSSI.

O.HASH The TOE shall check the sensitive data integrity with cryptographic keys different according to the container. The TOE shall generate these keys according to requirements for the standard robustness level of the cryptographic referential [CRYPTO_STD] and [CLES_STD] of ANSSI.

O.KEYS_CLEANING The TOE shall ensure the cleaning of sensitive data (encryption keys and elements of these keys) in the memory (RAM) at the end of every operation carried out by the TOE.

O.ALGO_STD The TOE shall provide random numbers and a choice of cryptographic algorithms and key sizes consistent with the state of the art and standards in this field, provided in [CRYPTO_STD] and supplemented by [CLES_STD] and

[AUTH_STD].

4.1.3. Container management

O.ADM_ACCESS The TOE shall provide an interface to the security administrator and to the access administrator to view accesses and to manage access keys to "containers" (Zed! Standard Edition only). Users can only view the accesses.

O.RECOVERY The TOE shall provide a mechanism to apply recovery and SOS access keys (Zed! Standard Edition only).

4.1.4. Protections when executing

O.POLICIES_INT The TOE shall verify the signature of new security policies to be applied (Zed! Standard Edition only). In case of failure, applied policies remain unchanged.

4.2. Security objectives for the operational environment

4.2.1. TOE use

OE.NON_OBSERV The user shall access his/her sensitive data only when he/she is in a reliable environment (when he/she is alone or with people that also need to know). Passwords shared between correspondents must be exchanged via secure organizational channels.

OE.OPERATIONAL_ENV When the user has been authenticated, the operational environment must guarantee the confidentiality of the sensitive data and the authentication data.

Application note: The hardware must ensure effective protection against

	eavesdropping and unauthorized data transmission (correctly configured firewall, up-to-date antivirus software, anti-spyware, etc.). Applications installed on the device must not interfere with the correct running of the TOE.
OE.TIMESTAMPING	The user must regularly check the workstation's clock to ensure the good quality of timestamping functions.
OE.ENV_RNG	The operating environment provides data so that the TOE implements mechanisms providing random numbers necessary for the generation of the secrets.

4.2.2. Users and administrator training

OE.TRAINING	The access administrator and the TOE users must be trained on TOE use and on the importance of IT security (this includes building awareness regarding the quality of access keys and their medium when kept in a key holder). The security administrator must receive training tailored to this function.
OE.EXT_CRYPTO	The security administrator must be aware of the importance placed on the quality of the access keys they provide to the TOE so that they comply with the state of the art as far as their implementation is concerned. He/she must also be aware of the importance of the quality of the medium for these keys when they are kept in an external key holder.
OE.KEY_STORAGE	Users are responsible for the safekeeping in a secure location and for the non-disclosure of access keys that are sent to them by a TOE administrator. The security administrator is responsible for the safekeeping in a secure location and for the non-disclosure of the recovery access keys.

4.2.3. Administration

OE.TRUST_ADMIN	The access and security administrators must be trustworthy. Windows administrators are trustworthy people responsible for the configuration (with secure values) of "policies". If the correspondents belong to entities managed by different security administrators, the latter must conjointly guarantee the use of security policies in compliance with the requisite levels (in particular, password strength).
OE.CERTIFICATES	When access keys that possess an X509 certificate are supplied, the security administrator must verify that these certificates are in fact valid and suitable for TOE usage. This requirement especially applies to root certificates called "authenticode", which can be used to verify TOE integrity.
OE.ENV_PROTECT_TOE	The technical environment of the TOE ensures the integrity of the TOE components and particularly its programs. The TOE administration and update are carried out by authorized administrators.

5. Security requirements

5.1. TOE security requirements

Each operation performed on components (assignment, selection, iteration and refinement) are identified with bold characters.

5.1.1. TOE Security functional requirements

Les composants fonctionnels sélectionnés pour répondre aux objectifs de sécurité de la TOE sont les suivants :

Selected CC components	
FCS_CKM.1	Cryptographic key generation
FCS_CKM.3	Cryptographic key access
FCS_CKM.4	Cryptographic key destruction
FCS_COP.1	Cryptographic operation
FDP_ACC.1	Subset access control
FDP_ACF.1	Security attribute based access control
FDP_ITC.1	Import of user data without security attributes
FDP_RIP.1	Subset residual information protection
FDP_SDI.2	Stored data integrity monitoring and action
FIA_AFL.1	Authentication failure handling
FIA_UAU.2	User authentication before any action
FIA_UID.2	User identification before any action
FMT_MOF.1	Management of security functions behaviour
FMT_MSA.1	Management of security attributes
FMT_MSA.2	Secure security attributes
FMT_MSA.3	Static attribute initialisation
FMT_MTD.1	Management of TSF data
FMT_SMF.1	Specification of Management Functions
FMT_SMR.1	Security roles

Table 2 : TOE functional security requirement

5.1.1.1 Introduction

The functional security requirements refer to the following subjects:

- Security administrator, access administrator and TOE users with their role and their access key (allowing to perform or not perform operations on containers) as security attributes.

The functional security requirements refer to the following objects:

- Containers handled by the TOE users and that contain user sensitive data (files, keys).

The functional security requirements refer to the following operations:

- Container management (policies, recovery and SOS procedure, access management)
- Use of containers

5.1.1.2 Class FCS : Cryptographic support

FCS_CKM	**Cryptographic key management**
FCS_CKM.1	Cryptographic key generation
FCS_CKM.1.1	The TSF shall generate cryptographic keys in accordance with a specified cryptographic key generation algorithm [**pseudo_random numbers generation and key diversification**] and specified cryptographic key sizes [**128, 192 and 256 bits for symmetric keys and 2048 bits for asymmetric keys**] that meet the following: [**ANSSI cryptographic requirements defined in [CRYPTO_STD] and [CLES_STD]**]. **Non editorial refinement:** **This component is related to the standard Edition only (the limited Edition only performs the key diversification to get the user access key from his/her password).**
FCS_CKM.3	Cryptographic key access
FCS_CKM.3.1	The TSF shall perform [**keys use**] in accordance with a specified cryptographic key access method [**keyboard driver and key decryption (unwrapping) by the access key**] that meets the following: [**None**]. **Non editorial refinement :** **This component is entirely implemented by the standard edition in charge of all sensitive administrative operations. But the limited Edition does not have any keyboard driver**

	and uses the key decryption only.
FCS_CKM.4	Cryptographic key destruction
FCS_CKM.4.1	The TSF shall destroy cryptographic keys in accordance with a specified cryptographic key destruction method [**wiping with random patterns**] that meets the following: [**none**].
FCS_COP	**Cryptographic operation**
FCS_COP.1	Cryptographic operation
FCS_COP.1.1	The TSF shall perform [**hash, encryption, decryption, signature verification of security policies, key wrapping and key derivation**] in accordance with a specified cryptographic algorithm [**HMAC, SHA-1 (key derivation), SHA-256, RSA, AES**] and cryptographic key sizes [**128, 192 and 256 bits for symmetric keys and 2048 bits for asymmetric keys**] that meet the following: [**ANSSI cryptographic requirements defined in [CRYPTO_STD] and [CLES_STD]**]. **Non editorial refinement:** **The Light Edition only implements file encryption and decryption so as to key derivation with the associated hash function.**

5.1.1.3 Class FDP: User data protection

FDP_ACC	**Access control policy**
FDP_ACC.1	Subset access control
FDP_ACC.1.1	The TSF shall enforce the [**SFP.ACCESS_OBJ**] on [**Subjects : Administrator and TOE users** **Objects : Container with user files and the control file** **Operations: Container management and use**].
FDP_ACF	**Access control functions**
FDP_ACF.1	Security attribute based access control

FDP_ACF.1.1	The TSF shall enforce the [**SFP.ACCESS_OBJ**] to objects based on the following: [**Subjects** : **Administrator and TOE users** **Security attributes: User access keys (allowing to open or not the containers) and role**]. **Non editorial refinement :** **The light Edition only deals with user role (no administrator role in this edition).**
FDP_ACF.1.2	The TSF shall enforce the following rules to determine if an operation among controlled subjects and controlled objects is allowed: [**Object : Container** **Operation: Container management and use** **Rule: successful authentication after access key input for the associated container with access to the container management for the access administrator role (access management only) and for the security administrator role (all management operations)**].
FDP_ACF.1.3	The TSF shall explicitly authorise access of subjects to objects based on the following additional rules: [**Aucune**].
FDP_ACF.1.4	The TSF shall explicitly deny access of subjects to objects based on the following additional rules: [**Aucune**].

FDP_ITC	**Import from outside TSF control**
FDP_ITC.1	Import of user data without security attributes
FDP_ITC.1.1	The TSF shall enforce the [**SFP.ACCESS_OBJ and SFP.ACCESS_ROLES**] 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**].

FDP_RIP	**Residual information protection**
FDP_RIP.1	Subset residual information protection
FDP_RIP.1.1	The TSF shall ensure that any previous information content of a resource is made unavailable upon the [**deallocation of the resource from**] the following objects: [**Container encryption keys and access keys**].

FDP_SDI	**Stored data integrity monitoring and action**
FDP_SDI.2	Stored data integrity monitoring and action
FDP_SDI.2.1	The TSF shall monitor user data stored in containers controlled by the TSF for [**integrity errors**] on all objects, based on the following attributes: [**HMAC for each file calculated from the content and some technical properties of these files**].
FDP_SDI.2.2	Upon detection of a data integrity error, the TSF shall [**display an error message**]. **Non editorial refinement :** **The control is performed as soon as the file is opened.**

5.1.1.4 Class FIA : Identification and authentication

FIA_AFL	**Authentication failures**
FIA_AFL.1	Authentication failure handling
FIA_AFL.1.1	The TSF shall detect when [**five**] unsuccessful authentication attempts occur related to [**container opening**].
FIA_AFL.1.2	When the defined number of unsuccessful authentication attempts has been met or surpassed, the TSF shall [**temporize the access to this container**].

FIA_UAU	**User authentication**
FIA_UAU.2	User authentication before any action
FIA_UAU.2.1	The TSF shall require each user to be successfully authenticated before allowing any other TSF-mediated actions on behalf of that user.

FIA_UID	**User identification**
FIA_UID.2	User identification before any action
FIA_UID.2.1	The TSF shall require each user to be successfully identified before allowing any other TSF-mediated actions on behalf of that user.

5.1.1.5 Class FMT : Security management

FMT_MOF	**Management of functions in TSF**
FMT_MOF.1	Management of security functions behaviour
FMT_MOF.1.1	The TSF shall restrict the ability to [**disable or enable**] the functions [**recovery and SOS]** to [**security administrator**].

	Non editorial refinement: **This component is related to the standard Edition only (the limited edition does not offer any management function).**
FMT_MSA	**Management of security attributes**
FMT_MSA.1	Management of security attributes
FMT_MSA.1.1	The TSF shall enforce the [**SFP.ACCESS_ROLES]** to restrict the ability to [**change_default, modify, delete**] the security attributes [**access keys and roles**] to [**access administrator and security administrator**]. **Non editorial refinement:** **This component is related to the standard Edition only (the limited edition does not offer any access management function).**
FMT_MSA.2	Secure security attributes
FMT_MSA.2.1	The TSF shall ensure that only secure values are accepted for [**access keys**]. **Non editorial refinement:** **This component is related to the standard Edition only (the limited edition does not offer any access management function).**
FMT_MSA.3	Static attribute initialisation
FMT_MSA.3.1	The TSF shall enforce the [**SFP.ACCESS_ROLES**] to provide [**restrictive**] default values for security attributes that are used to enforce the SFP.
FMT_MSA.3.2	The TSF shall allow the [**access administrator and security administrator**] to specify alternative initial values to override the default values when an object or information is created. **Non editorial refinement:** **This component is related to the standard Edition only (the limited edition does not offer any access management function).**
FMT_MTD	**Management of TSF data**
FMT_MTD.1	Management of TSF data
FMT_MTD.1.1	The TSF shall restrict the ability to [**change_default, modify, delete**] the [**policies**] to [**the security administrator**]. **Non editorial refinement:** **This component is related to the standard Edition only (the limited edition does not offer any management function).**
FMT_SMF	**Specification of Management Functions**

FMT_SMF.1	Specification of Management Functions
FMT_SMF.1.1	The TSF shall be capable of performing the following management functions: [- **Access management functions** - **Recovery and SOS functions]** **Non editorial refinement:** **This component is related to the standard Edition only (the limited edition does not offer any management function).**
FMT_SMR	**Security management roles**
FMT_SMR.1.	Security roles
FMT_SMR.1.1	The TSF shall maintain the roles [**security administrator, access administrator and TOE users**].
FMT_SMR.1.2	The TSF shall be able to associate users with roles.

5.1.2. TOE security assurance requirements

As indicated in chapter 3.3, the TOE must demonstrate resistance to penetration attackers with an Enhanced-Basic attack potential.

The target assurance level is the level:

EAL3 augmented with ALC_FLR.3 and AVA_VAN.3 components associated with analysis of the cryptographic implementation described in [QUALIF_STD].

This level corresponds to the selection of following assurance components:

Component		Comment
ADV_ARC.1	Security architecture description	EAL3
ADV_FSP.3	Functional specification with complete summary	EAL3
ADV_TDS.2	Architectural design	EAL3
AGD_OPE.1	Operational user guidance	EAL3
AGD_PRE.1	Preparative procedures	EAL3
ALC_CMC.3	Authorisation controls	EAL3
ALC_CMS.3	Implementation representation CM coverage	EAL3
ALC_DEL.1	Delivery procedures	EAL3
ALC_DVS.1	Identification of security measures	EAL3
ALC_FLR.3	Systematic flaw remediation	+
ALC_LCD.1	Developer defined life-cycle model	EAL3
ASE_CCL.1	Conformance claims	EAL3
ASE_ECD.1	Extended components definition	EAL3
ASE_INT.1	ST introduction	EAL3
ASE_OBJ.2	Security objectives	EAL3
ASE_REQ.2	Security requirements	EAL3
ASE_SPD.1	Security problem definition	EAL3
ASE_TSS.1	TOE summary specification	EAL3
ATE_COV.2	Analysis of coverage	EAL3
ATE_DPT.1	Testing: basic design	EAL3
ATE_FUN.1	Functional testing	EAL3
ATE_IND.2	Independent testing - sample	EAL3
AVA_VAN.3	Focused vulnerability analysis	+

Table 3 : Security assurance components

This assurance level complies with dependencies between CC assurance components mentioned in CC part 3.

6. TOE summary specifications

The security functions that are performed by the TOE are described in this chapter.

F.ACCESS_CONTROL **Access control to containers**

This security function is the interface enforcing the access control to open the containers controlled by the TOE. The TSF allows or denies access to an encrypted container on the basis of the verification of a "login / authentication" couple provided by the TOE user. A delay is applied after five consecutive failures.

F.SECURE_INPUT **secure input**

This security function covers the secure communication of input data using encryption and decryption functions of encryption keys and the keyboard driver (standard edition only) when a password or a PIN code of a key file is entered.

F.TOE_CONFIGURATION **Modifying the TOE configuration**

This security function covers all the TOE configuration operations (initialization and modification) and ensures that only secure values are accepted. Configuration data concern the Windows policies which are signed by the security administrator and operated by the TOE (after successful verification of the signature). These configuration data define the access types supported, the cryptographic algorithms (AES 256-bit by default), the password complexity, the control of the certificates etc… If the verification is correct, the new policies are applied on the workstation.

F.ACCESS_KEY_MANAGEMENT **Key management**

This security function manages the security attributes: access keys and associated roles (user, security administrator or access administrator). An access corresponds to an access key (a cryptographic key) that a user owns in order to obtain the encryption / decryption elements of the container. If these elements are extracted for access management operations, the input access key must be associated with the access (or security) administrator role. This function also manages the recovery access which is a particular access and the SOS key .

The F.KEY_MANAGEMENT function also performs RSA key generation of access lists, access key addition and suppression (standard edition) as well as the cryptographic key accesses (in particular via the pkcs#11 token). It provides the secure cleaning of these keys in memory after a session locking or sleep mode (standard edition), a session logout or a system logoff.

F.CRYPTO_OPERATIONS **Cryptographic operations implementation**

This security feature covers the encryption and HMAC key generations the file encryption and decryption in the container, the operations related to the data integrity verification and all the cryptographic operations used by other security features.

Some cryptographic operations (key generation for example) are not available in the limited edition.

7. Protection profile conformance

This security target does not claim any compliance with a protection profile.

8. Rationale

8.1. Security objectives rationale

This section provides the mappings between security objectives and the elements which constitute the definition of the TOE environment (assumptions, organizational policies and threats).

8.1.1. Assumptions

The table below presents the coverage of the selected assumptions by the security objectives:

		OE.NON_OBSERV	OE.OPERATIONNAL_ENV	OE.TIMESTAMPING	OE.TRUST_ADMIN	OE.KEY_STORAGE	OE.CERTIFICATES	OE.ENV_PROTECT_TOE	OE.TRAINING	OE.ENV_RNG	OE.EXT_CRYPTO
Assomptions	A.NON_OBSERV	X									
	A.SECURE_PC		X								
	A.TRUST_ADMIN				X				X		
	A.KEY_STORAGE					X			X		
	A.CERTIFICATES						X				
	A.ENV_PROTECT_TOE							X	X		
	A.LOYAL_ENV			X							
	A.ENV_RNG									X	
	A.EXT_CRYPTO										X

Table 4 : Assumptions to security objectives mapping

A.NON_OBSERV	The physical environment for the use of the TOE enables users to enter their passwords (or PIN numbers) without being directly observable and protecting them from other users or hackers to intercept their passwords or PIN numbers (wireless keyboard, etc.).

The OE.NON_OBSERV objective addresses directly this assumption in providing the user for an adequate environment.

A.SECURE_PC	The operational environment does not allow any attacker to access the hard disk when sensitive data are processed on the workstation by an authorized user. The user workstation must ensure effective protection against eavesdropping and unauthorized data transmission (correctly configured firewall, up-to-date antivirus software, anti-spyware, etc.).

The OE.OPERATIONNAL_ENV objective addresses directly this assumption in providing the user for an adequate environment.

A.TRUST_ADMIN	The security administrator and the access administrator are trustworthy. Windows administrators are trustworthy people responsible for the policy configuration (with secure values). All these people and the users are trained to the TOE usage. If the correspondents belong to entities managed by different Windows environment security administrators, the latter must conjointly guarantee the use of security policies in compliance with the requisite levels (in particular, password strength).

The OE.TRUST_ADMIN and OE.TRAINING objectives address this assumption in employing and training trustworthy people.

A.KEY_STORAGE	Users are responsible for the safekeeping in a secure location and for the non-disclosure of access keys that are sent to them by a TOE administrator. The TOE administrator is responsible for the safekeeping in a secure location and for the non-disclosure of the recovery access keys and his/her signature key.

The OE.KEY_STORAGE and OE.TRAINING objectives address this assumption in ensuring that users and administrators are accountable.

A.CERTIFICATES	When access keys that possess an X509 certificate are supplied, the TOE administrator must verify that these certificates are in fact valid and suitable for TOE usage.

The OE.CERTIFICATES objective addresses directly this assumption.

A.ENV_PROTECT_TOE	The technical environment of the TOE ensures the integrity of the TOE components. The TOE administration and update are carried out by traine and authorized people.

The OE.ENV_PROTECT_TOE and OE.TRAINING objectives address this assumption in ensuring that the programs are not illegally modified and administrators are trained.

A.LOYAL_ENV	The runtime environment provides the TOE with exact date and time to ensure time stamp functions.

The OE.TIMESTAMPING objective addresses directly this assumption.

A.ENV_RNG	The TOE implements mechanisms to provide the random numbers necessary for the generation of the secrets.

The OE.ENV_RNG objective addresses directly this assumption.

A.EXT_CRYPTO	The access keys generated or stored outside the TOE must be compliant with [CRYPTO_STD] document for the Standard robustness level.

The OE.EXT_CRYPTO objective addresses directly this assumption.

8.1.2. Threats

The table below presents the coverage of the selected threats by the security objectives:

		O.AUTH	O.ROLES	O. ENCRYPTION	O.HASH	O. KEYS_CLEANING	O.ALGO_STD	O.ADM_ACCES	O.RECOVERY	O.POLICIES_INT	OE.TRUST_ADMIN	OE. KEY_STORAGE
Threats	T.COMPONENT_MISUSE	X				X	X					
	T.POLICIES_SECU_INT									X	X	X
	T.CONTROL_FILE_CONF	X		X			X					
	T.CONTROL_FILE_INT	X		X	X		X					
	T.CATALOGUE_FILE_INT	X		X	X		X					

Table 5 : Threats to security objectives mapping

T.COMPONENT_MISUSE

A person intercepts a container, get the Zed! product (or Zed! Limited Edition) and manipulates, possibly at low level, internal components of the TOE, to bypass some security functions by **causing or taking advantage of a malfunction**. He can do this by reverse-engineering programs, or by developing programs calling internal functions of the TOE, or by modifying the internal configuration of the TOE or by using a debugger. Impacted asset is the TOE program (confidentiality and integrity) and the configuration (integrity).

By these different means, the attacker must not succeed to "enter" a container in which he/she has not access.

➔ To prevent the threat, the TOE must:

- Ensure that, before any operation on the TOE, authentication is necessary (O.AUTH).

➔ To protect itself, the TOE must:

- Ensure that it is not cryptographically possible to find out encryption keys of the container without providing a valid access key: the misuse of a TOE component does not allow to bypass this protection (O.AUTH and O.ALGO_STD).
- Ensure that a misused component does not retain any residues providing a possible attack path (O.KEYS_CLEANING).

➔ To limit the threat impact the TOE must:

None

T.POLICIES_SECU_INT

A person taking advantage of a malfunction signs the policies in place of the security administrator (domain policies areas or local policies if the attacker can access the workstation). For example, he/she can configure his/her own recovery access which will be added automatically when the user will create the next containers. The impacted asset is the configuration (integrity).

➔ To prevent the threat, the TOE must:

- Ensure that administrators (including the Windows administrators) are trustworthy people (OE. TRUST_ADMIN);
- Ensure that the security administrator keeps his signature private key in a safe place (OE. KEYS_STORAGE)

➔ To protect itself, the TOE must:

- Ensure that it is not possible to apply security policies (and modify the related configuration file) without policies signature by the private key of the security officer (O.POLICIES_INT)

➔ To limit the threat impact the TOE must:

None

T.CONTROL_FILE_CONF

An eavesdropper or an unauthorized user retrieves the control file of the TOE in an attempt to find protected information. The impacted asset is therefore the control file of the container (confidentiality).

For example, the attacker tries to find some protected information (e.g. encryption keys) from the encrypted files of the container and from the TOE control file or tries to decrypt (brute force) the information stored in the control file.

➔ To prevent the threat, the TOE must:

- Ensure that, before any operation on the TOE, authentication is necessary (O.AUTH).

➔ To protect itself, the TOE must:

- Ensure that it is not cryptographically possible to find out encryption keys of the container without providing a valid access key, and that the control file enforces this requirement (O.AUTH and O.ALGO_STD).

➔ To limit the threat impact the TOE must:

- Ensure that the internal control files are "cryptographically different" by using random numbers that does not allow to get information from a control file in order to attack another one (O.ENCRYPTION and O.ALGO_STD).

T.CONTROL_FILE_INT	**An eavesdropper or an unauthorized user** retrieves the control file of the TOE (stored in the container) and modifies it in an attempt to add its own access among authorized accesses (the attacker can position itself between two correspondents for example). The impacted assets is therefore the control file of the TOE (integrity). The attacker can thus intercept and read the files exchanged between legitimate correspondents or send the container to a correspondent (by usurping the identity of a legitimate user) so that the correspondent send him sensitive files.

➔ To prevent the threat, the TOE must:

- Ensure that, before any operation on the TOE, authentication is necessary (O.AUTH).

➔ To protect itself, the TOE must:

- Ensure that it is not possible, cryptographically, to find out container encryption keys without providing a valid access key : the unauthorized modification of the control file is prohibited thanks to this mechanism (O.AUTH and O.ALGO_STD);
- Ensure that any integrity violation of the control file will be detected and notified to the user who opens the container (O.HASH).

➔ To limit the threat impact the TOE must:

- Ensure that the internal files of the different containers are "cryptographically different" by using random numbers that does not allow to get information from a control file in order to attack another one (O.ENCRYPTION and O.ALGO_STD).

T.CATALOGUE_FILE_INT	**An eavesdropper or an unauthorized user** retrieves the catalogue file of the TOE (stored in the container) and modifies it in an attempt to

	mofigy the container tree view. The impacted assets is therefore the catalogue file of the TOE (integrity). The attacker can thus intercept the exchanged containers and make disappear one or more files in the container without any detection by the recipient.

➔ To prevent the threat, the TOE must:

- Ensure that, before any operation on the TOE, authentication is necessary (O.AUTH).

➔ To protect itself, the TOE must:

- Ensure that it is not possible, cryptographically, to find out container encryption keys without providing a valid access key : the unauthorized modification of the catalogue file is prohibited thanks to this mechanism (O.AUTH and O.ALGO_STD);
- Ensure that any integrity violation of the catalogue file will be detected and notified to the user who opens the container (O.HASH).

➔ To limit the threat impact the TOE must:

- Ensure that the internal files of the different containers are "cryptographically different" by using random numbers that does not allow to get information from a catalogue file in order to attack another one (O.ENCRYPTION and O.ALGO_STD).

8.1.3. Organizational security policies

The table below presents the coverage of the selected organizational security policies by the security objectives:

		O.AUTH	O.ROLES	O.ENCRYPTION	O.HASH	O.KEYS_CLEANING	O.ALGO_STD	O.ADM_ACCESS	O.RECOVERY	O.POLICIES_INT
OSP	OSP.CONFIDENTIALITY			X		X	X			
	OSP.INTEGRITY				X	X	X			
	OSP.ACCESS	X				X				
	OSP.RECOVERY	X	X			X			X	
	OSP.ADMIN_ACCESS	X	X			X		X		
	OSP.POLICIES_VERIF									X
	OSP.CRYPTO						X			

Table 6 : Organizational security policies to security objectives mapping

---

OSP.CONFIDENTIALITY The TOE shall offer automatic and systematic confidentiality protection (encryption) of user sensitive files that are stored or send (email attachment).

Note: This policy concerns the initial creation of the container, and the fact that once the container is created, any file copied into it is stored encrypted. This policy does not concern accesses to the container, which are covered by OSP.ACCESS.

➔ To cover this policy, the TOE must:

- Generates random numbers to create the encryption key of the container (O.ALGO_STD);
- Encrypt the files in the container (O.ENCRYPTION) ;

➔ To ensure the implementation of the policy, the TOE must:

- Erase the residual information in memory (information related to encryption keys) (O.KEYS_CLEANING).

OSP.INTEGRITY	The TOE shall provide automatic and systematic integrity check (hash) of user sensitive files in the containers.

➔ To cover this policy, the TOE must:

- Generates random numbers to create the HMAC key of the container (O.ALGO_STD);
- Calculate a MAC and an HMAC for each file of the container (O.HASH);
- Compute an HMAC in the catalogue file (O.HASH) to detect the deletion of a file by an attacker.

➔ To ensure the implementation of the policy, the TOE must:

- Erase the residual information in memory (information related to HMAC keys) (O.KEYS_CLEANING).

OSP.ACCESS	The TOE shall enable users to provide an access key allowing the access to sensitive files of the container in which they wish to access. If they cannot provide a valid key for the container, access must be rejected.

Note: This policy does not concern the access management (adding or deleting an access enforced by OSP.ADMIN_ACCES), but the use of an access.

➔ To cover this policy, the TOE must:

- Require each user to be successfully authenticated before accessing a file in the container (O.AUTH);

➔ To ensure the implementation of the policy, the TOE must:

- Ensure that only a valid access key is able to recover the container encryption key (using TOE internal information and internal files (O.AUTH).
- Erase the residual information in memory that is related to intermediary cryptographic calculations (password derivation) or to the transfer of encryption keys when they are calculated by an external cryptographic device (token) (O.KEYS_CLEANING).

OSP.RECOVERY	The TOE shall offer a service to recover sensitive files of users by using recovery access keys managed by the security administrator. These keys are systematically and automatically assigned during container initialization. The TOE must also provides a remote recovery (SOS procedure) if the user forgot his/her password or lost/broken his/her token. This SOS procedure is performed with a key systematically and automatically assigned during the creation of the user access list. This policy applies only to Zed! Standard Edition.

➔ To cover this policy, the TOE must:

- Offers a service to assign recovery and SOS access keys to the container (O.RECOVERY)
- Require authentication to access the management of the recovery and the SOS keys (O.AUTH);
- Ensure that the security administrator is the only authorized person performing recovery and SOS operations (O.ROLES)

➔ To ensure the implementation of the policy, the TOE must:

- Ensure that only a valid access key is able to recover the container encryption key (using TOE internal information and internal files (O.AUTH).
- Erase the residual information in memory that is related to intermediary cryptographic calculations (password derivation) or to the transfer of encryption keys when they are calculated by an external cryptographic device (token) (O.KEYS_CLEANING).

OSP.ADMIN_ACCESS	The TOE shall offer a service to manage accesses (Zed! Standard Edition only).

➔ To cover this policy, the TOE must:

- Require the administrator to be successfully authenticated before performing any access management on encrypted container (O.AUTH) ;
- Offer an interface to the administrator so that he/she can visualize (the user can do it also) and manage the container access keys (O.ADM_ACCESS).
- Control that only a user with 'security or access administrator' role for a given can manage the accesses (O.ROLES) ;

➔ To ensure the implementation of the policy, the TOE must:

- Erase the residual information in memory (information related to access keys) (O.KEYS_CLEANING).

OSP.POLICIES_VERIF	The TOE shall provide a service without (special handling by the user) that performs the verification of the signature of security policies signed by the security administrator's private key (Zed! Standard Edition only). The application of any new policy is conditioned by the success of this verification.

➔ To cover this policy, the TOE must:

- Requires successful signature verification of new policies in order to apply them (O.POLICIES_INT).

OSP.CRYPTO	The ANSSI referential ([CRYPTO_STD], [CLES_STD] and [AUTH_STD]) defined for the 'standard' robustness level must be applied for the key management and for the cryptographic and authentication mechanisms used in the TOE.

➔ To cover this policy, the TOE must:

- Provide a choice of cryptographic algorithms and key sizes consistent with the state of the art and standards in this field (O.ALGO_STD)
- Provide a choice of cryptographic algorithms and key sizes consistent with the state of the art and standards in this field, provided for in [CRYPTO_STD] and supplemented by [CLES_STD] for the random number generation and key generation) (O.ALGO_STD).

8.1.4. Summary on the coverage of the objectives

The table below presents a summary of the coverage of the security objectives by the assumptions, threats and organizational security policies:

		A.NON_OBSERV	A. SECURE _PC	A.TRSUT_ADMIN	A. KEY_STORAGE	A.CERTIFICATES	A.ENV_PROTECT_TOE	A.LOYAL_ENV	A.ENV_RNG	A.EXT_CRYPTO	T.COMPONENT_MISUSE	T.POLICIES_SECU_INT	T.CONTROL_FILE_CONF	T.CONTROL_FILE_INT	T.CATALOGUE_FILE_INT	OSP.CONFIDENTIALITY	OSP.INTEGRITY	OSP.ACCESY	OSP.RECOVERY	OSP.ADMIN_ACCESS	OSP.POLICIES_VERIF	OSP.CRYPTO
Security objectives	OE.NON_OBSERV	X																				
	OE.OPERATIONAL_ENV		X																			
	OE.TIMESTAMPING							X														
	OE.TRUST_ADMIN			X								X										
	OE.KEY_STORAGE				X							X										
	OE.CERTIFICATES					X																
	OE.ENV_PROTECT_TOE						X															
	OE.TRAINING			X	X		X															
	OE.ENV_RNG								X													
	OE.EXT_CRYPTO									X												
	O.AUTH										X		X	X	X			X	X	X		
	O.ROLES																		X	X		
	O.ENCRYPTION												X	X	X	X						
	O.HASH													X	X		X					
	O.KEYS_CLEANING										X					X	X	X	X	X		
	O.ALGO_STD										X		X	X	X	X	X					X
	O.ADM_ACCESS																			X		
	O.RECOVERY																		X			
	O.POLICIES_INT											X									X	

Table 7 : Coverage of the security objectives by the assumptions, threats and organizational security policies

8.2. Security requirements rationale

8.2.1. Functional security requirements dependencies

The table below presents the coverage of the dependencies between selected functional components:

Component	CC-required dependencies	Fulfilled dependencies
FCS_CKM.1	[FCS_CKM.2 or FCS_COP.1], FCS_CKM.4	FCS_COP.1, FCS_CKM.4
FCS_CKM.3	[FDP_ITC.1 or FDP_ITC.2 or FCS_CKM.1], FCS_CKM.4	FDP_ITC.1, FCS_CKM.1, FCS_CKM.4
FCS_CKM.4	[FDP_ITC.1 or FDP_ITC.2 or FCS_CKM.1]	FDP_ITC.1, FCS_CKM.1
FCS_COP.1	[FDP_ITC.1 or FDP_ITC.2 or FCS_CKM.1], FCS_CKM.4	FDP_ITC.1, FCS_CKM.1, FCS_CKM.4
FDP_ACC.1	FDP_ACF.1	FDP_ACF.1
FDP_ACF.1	FDP_ACC.1, FMT_MSA.3	FDP_ACC.1, FMT_MSA.3
FDP_ITC.1	[FDP_ACC.1 or FDP_IFC.1], FMT_MSA.3	FDP_ACC.1, FMT_MSA.3
FDP_RIP.1	None	None
FDP_SDI.2	None	None
FIA_AFL.1	FIA_UAU.1	FIA_UAU.2
FIA_UAU.2	FIA_UID.1	FIA_UID.2
FIA_UID.2	None	None
FMT_MOF.1	FMT_SMF.1, FMT_SMR.1	FMT_SMF.1, FMT_SMR.1
FMT_MSA.1	[FDP_ACC.1 or FDP_IFC.1], FMT_SMF.1, FMT_SMR.1	FDP_ACC.1, FMT_SMF.1, FMT_SMR.1
FMT_MSA.2	[FDP_ACC.1 or FDP_IFC.1], FMT_MSA.1, FMT_SMR.1	FDP_ACC.1, FMT_MSA.1, FMT_SMR.1
FMT_MSA.3	FMT_MSA.1, FMT_SMR.1	FMT_MSA.1, FMT_SMR.1
FMT_MTD.1	FMT_SMR.1, FMT_SMF.1	FMT_SMR.1, FMT_SMF.1
FMT_SMF.1	None	None
FMT_SMR.1	FIA_UID.1	FIA_UID.2

Table 8 : Functional security requirements dependencies

8.2.2. Assurance security requirements dependencies

The table below presents the coverage of the dependencies between selected assurance components:

Component	CC-required Dependencies	Fulfilled dependencies
ADV_ARC.1	ADV_FSP.1, ADV_TDS.1	ADV_FSP.3, ADV_TDS.2
ADV_FSP.3	ADV_TDS.1	ADV_TDS.2
ADV_TDS.2	ADV_FSP.3	ADV_FSP.3
AGD_OPE.1	ADV_FSP.1	ADV_FSP.3
AGD_PRE.1	None	None
ALC_CMC.3	ALC_CMS.1, ALC_DVS.1, ALC_LCD.1	ALC_CMS.3, ALC_DVS.1, ALC_LCD.1
ALC_CMS.3	None	None
ALC_DEL.1	None	None
ALC_DVS.1	None	None
ALC_FLR.3	None	None
ALC_LCD.1	None	None
ASE_CCL.1	ASE_INT.1, ASE_ECD.1, ASE_REQ.1	ASE_INT.1, ASE_ECD.1, ASE_REQ.2
ASE_ECD.1	None	None
ASE_INT.1	None	None
ASE_OBJ.2	ASE_SPD.1	ASE_SPD.1
ASE_REQ.2	ASE_OBJ.2, ASE_ECD.1	ASE_OBJ.2, ASE_ECD.1
ASE_SPD.1	None	None
ASE_TSS.1	ASE_INT.1, ASE_REQ.1, ADV_FSP.1	ASE_INT.1, ASE_REQ.2, ADV_FSP.3
ATE_COV.2	ADV_FSP.2, ATE_FUN.1	ADV_FSP.3, ATE_FUN.1
ATE_DPT.1	ADV_ARC.1, ADV_TDS.2, ATE_FUN.1	ADV_ARC.1, ADV_TDS.2, ATE_FUN.1
ATE_FUN.1	ATE_COV.1	ATE_COV.2
ATE_IND.2	ADV_FSP.2, AGD_OPE.1, AGD_PRE.1, ATE_COV.1, ATE_FUN.1	ADV_FSP.3, AGD_OPE.1, AGD_PRE.1, ATE_COV.2, ATE_FUN.1
AVA_VAN.3	ADV_ARC.1, ADV_FSP.2, ADV_TDS.3*, ADV_IMP.1*, AGD_OPE.1, AGD_PRE.1	ADV_ARC.1, ADV_FSP.3, AGD_OPE.1, AGD_PRE.1

Table 9 : Assurance security requirements dependencies

8.2.3. Justification of unsupported dependencies

* AVA_VAN.3 dependency with ADV_FSP.4, ADV_IMP.1 and ADV_TDS.3 are not fulfilled in accordance with the assurance package relative to qualification process at the "standard" robustness level defined by ANSSI [QUALIF_STD].

8.2.4. Security objectives to functional security requirements mapping rationale

The table below presents the coverage of the security objectives by the selected functional security requirements:

Objectifs de sécurité de la TOE	FCS_CKM.1	FCS_CKM.3	FCS_CKM.4	FCS_COP.1	FDP_ACC.1	FDP_ACF.1	FDP_ITC.1	FDP_RIP.1	FDP_SDI.2	FIA_AFL.1	FIA_UAU.2	FIA_UID.2	FMT_MOF.1	FMT_MSA.1	FMT_MSA.2	FMT_MSA.3	FMT_MTD.1	FMT_SMF.1	FMT_SMR.1
O.AUTH					X	X	X			X	X	X							
O.ROLES					X	X							X				X	X	X
O. ENCRYPTION	X	X		X															
O.HASH	X	X		X					X										
O.KEYS_CLEANING								X											
O.ALGO_STD	X	X	X	X															
O.ADM_ACCES														X	X	X		X	X
O.RECOVERY													X				X	X	X
O.POLICIES_INT				X													X		

Table 10 : Security objectives to functional security requirements mapping

8.2.4.1 Access control

O.AUTH	The TOE shall identify and authenticate users. To do this, the TOE shall allow access to a container only after presentation of a valid key for the container.

In order to fulfill this objective:

- The TOE identifies and authenticates each user before allowing any operation (FIA_UAU.2 and FIA_UID.2) and slowdowns the display of the authentication window after several unsuccessful authentication attempts (FIA_AFL.1).
- In order to access to an encrypted container, the user must enter his/her access keys (USB token for example) to be authenticated (FDP_ITC.1).
- Then the TOE enforces an access control policy to the container (FDP_ACC.1) based on security attributes (FDP_ACF.1).

O.ROLES	The TOE shall manage three user roles in an encrypted container: a 'normal user' role or simply 'user' role (use of files in the container after presentation of a valid access key), a ‘access administrator’ role in charge of initializing the container and adding the authorized accesses and a 'security administrator' role (installation, policies signature, recovery, access management).

In order to fulfill this objective:

- The TOE must manage and differentiate the security administrator, access administrator and user roles (FMT_SMR.1).
- The TOE also controls the user access to containers and to the operations on containers (FDP_ACC.1), and restricts the access to the users having a valid access key (FDP_ACF.1).
- Finally the TOE must restrict the ability to manage security functions (FMT_SMF.1), policies (FMT_MTD.1) and recovery (FMT_MOF.1) to the administrator.

8.2.4.2 Cryptography

O.ENCRYPTION	The TOE shall encrypt and decrypt sensitive data with cryptographic keys. The TOE shall use different encryption keys to protect the different containers. The TOE shall generate encryption keys according to requirements for the standard robustness level of the cryptographic referential [CRYPTO_STD] and [CLES_STD] of ANSSI.

In order to fulfill this objective:

- In order to encrypt the files in the container, the TOE must be able to generate cryptographic keys (FCS_CKM.1) and to access them in a secure way (FCS_CKM.3), to use them and perform the cryptographic operations in accordance with different algorithms (FCS_COP.1).

O.HASH	The TOE shall check the sensitive data integrity with cryptographic keys different according to the container. The TOE shall generate these keys according to requirements for the standard robustness level of the cryptographic referential [CRYPTO_STD] and [CLES_STD] of ANSSI.

In order to fulfill this objective:

- In order to calculate the HMAC of the user files, control file and catalogue file of the container, the TOE must be able to generate cryptographic keys (FCS_CKM.1) and to access them in a secure way (FCS_CKM.3), to use them and perform the cryptographic operations in accordance with different algorithms (FCS_COP.1).
- The TOE check the integrity of sensitive files stored in the containers and returns a warning in case of error detection (FDP_SDI.2).

O.KEYS_CLEANING	The TOE shall ensure the cleaning of sensitive data (encryption keys and elements of these keys) in the memory (RAM) at the end of every operation carried out by the TOE.

In order to fulfill this objective:

- The TOE ensures the secure cleaning of keys in the RAM memory (FDP_RIP.1).

O.ALGO_STD	The TOE shall provide random numbers and a choice of cryptographic algorithms and key sizes consistent with the state of the art and standards in this field, provided in [CRYPTO_STD] and supplemented by [CLES_STD] and [AUTH_STD].

In order to fulfill this objective:

- The TOE must be able to implement a method of cryptographic key generation (FCS_CKM.1), cryptographic key access (FCS_CKM.3) and cryptographic key destruction (FCS_CKM.4).
- The TOE must perform cryptographic operations in accordance with algorithms and key sizes specified in (FCS_COP.1).

8.2.4.3 Container management

O.ADM_ACCESS	The TOE shall provide an interface to the security administrator and to the access administrator to view accesses and to manage access keys to "containers" (Zed! Standard Edition only). Users can only view the accesses.

In order to fulfill this objective:

- The TOE offers functions to manage the accesses (FMT_SMF.1)

- The TOE gives access to management function based on the role associated with users (FMT_SMR.1)
- The TOE ensures that only security and access administrators are allowed to manage the security attributes of stored objects: keys and roles (FMT_MSA.1).
- The administrators can also define the initialization values of the attributes (such as the role that is initialized by default to « user ») (FMT_MSA.3).
- The TOE ensures that only secure values are accepted for security attributes (minimum password strength for example) (FMT_MSA.2).

O.RECOVERY	The TOE shall provide a mechanism to apply recovery and SOS access keys (Zed! Standard Edition only).

In order to fulfill this objective:

- The TOE shall restrict the ability to disable or enable the recovery function (FMT_SMF.1) to the security administrator (FMT_MOF.1).
- The recovery function is configured in the policies signed by the security administrator (FMT_MTD.1)

8.2.4.4 Protections when executing

O.POLICIES_INT	The TOE shall verify the signature of new security policies to be applied (Zed! Standard Edition only). In case of failure, applied policies remain unchanged.

In order to fulfill this objective:

- The TOE must perform signature verification operations in accordance with algorithms and key sizes specified in (FCS_COP.1).
- The TOE shall verify that the signature operation was performed by the security administrator (FMT_MTD.1).

8.2.5. Rationale for Assurance Level 3 Augmented

The EAL3 assurance level augmented with ALC_FLR.3 and AVA_VAN.3 components associated with analysis of the cryptographic implementation was chosen to comply with qualification process at the "standard" robustness level defined by ANSSI [QUALIF_STD]. This assurance level imposes:

- Independent testing performed by the evaluator (the final user is then ensured that the TOE security functions are implemented as specified)
- Independent vulnerability analysis by the evaluator who will consider an attacker possessing an Enhanced-Basic (or inferior) attack potential (the final user is then ensured that the TOE is resistant to penetration attacks performed by attackers possessing an "enhanced-basic" attack potential).
- A security architecture and a software architecture including implementation analysis (cryptographic functions only) evaluation to verify any security malfunctions ;
- Software development good practices (the final user is then ensured that the cryptographic part of the product was correctly and securely designed and developed).
- Good practices in maintenance and user support ensuring that all identified flaws will be corrected and reported to the product users that might be affected by these anomalies.

8.3. Rationale for TOE summary specifications

The table below presents the coverage of the functional security requirements by the TOE security functions:

Functional security requirements for the TOE		F.ACCESS_CONTROL	F.SECURE_INPUT	F.TOE_CONFIGURATION	F.ACCESS_KEY_MANAGEMENT	F.CRYPTO_OPERATIONS
FCS_CKM.1	Cryptographic key generation				X	X
FCS_CKM.3	Cryptographic key access		X			
FCS_CKM.4	Cryptographic key destruction				X	
FCS_COP.1	Cryptographic operation	X	X	X	X	X
FDP_ACC.1	Subset access control	X			X	
FDP_ACF.1	Security attribute based access control	X			X	
FDP_ITC.1	Import of user data without security attributes		X			
FDP_RIP.1	Subset residual information protection				X	
FDP_SDI.2	Stored data integrity monitoring and action					X
FIA_AFL.1	Authentication failure handling	X				
FIA_UAU.2	Authentication failure handling	X	X	X		
FIA_UID.2	User authentication before any action	X	X	X		
FMT_MOF.1	User identification before any action	X		X		
FMT_MSA.1	Management of security functions behaviour				X	
FMT_MSA.2	Management of security attributes			X		
FMT_MSA.3	Secure security attributes				X	
FMT_MTD.1	Static attribute initialisation			X		
FMT_SMF.1	Management of TSF data			X	X	
FMT_SMR.1	Specification of Management Functions				X	

Table 11 : TOE summary specifications to functional security requirements mapping

FCS_CKM.1 Cryptographic key generation

A key is associated with each container to encrypt and decrypt the files and a HMAC key. These keys are generated when the container is initialized (standard edition only). The keys meet the algorithm and length criteria configured in the policies. By default, they are 256-bit AES keys.

The format of certain user access keys (personal access list) can also be subject to an intermediate encryption by a RSA key generated by the TOE.

The security functions F.ACCESS_KEY_MANAGEMENT implements the RSA key generation and F.CRYPTO_OPERATIONS the AES key generation.

FCS_CKM.3 Cryptographic key access

Cryptographic key access managed by the TOE is implemented by the security function F.SECURE_INPUT to secure input password and PIN codes.

This function is used by the user authentication process.

FCS_CKM.4 Cryptographic key destruction

When a container is deleted, the cryptographic keys used by the container are destroyed. Similarly when an access is removed (performed only by the standard edition), the corresponding access key is destroyed.

The security function F.ACCESS_KEY_MANAGEMENT enforces this functional requirement.

FCS_COP.1 Cryptographic operation

The TOE performs the following cryptographic operations:

- Gets an access key before being able to create a container key (initialization) and encrypt the container,
- Gets an access key to decrypt (unwrap) the container key and add a new access key. Then this access key wraps the container key,
- Gets an access key to decrypt the container key and encrypt or decrypt the container files,

- Gets an access key to decrypt the HMAC key and perform HMAC of the sensitive data,
- Gets a password to derive an access key and encrypt or decrypt the container key,
- Send the container encrypted key to the key holder, then get back the container key unencrypted by the key holder and decrypt the container files,
- Verifies the policies signature with the security officer certificate.

The security function F.CRYPTO_OPERATIONS implements the encryption and decryption operations, integrity control as well as cryptographic operations in the used by other functions.

The functions F.ACCESS_KEY_MANAGEMENT (access key creation) and F.ACCESS_CONTROL (access key verification) use the functions for deriving keys from passwords

The security function F.SECURE_INPUT uses wrapping functions to ensure the secure transfer of keys between the TOE and the key holders.

The security function F.TOE_CONFIGURATION is involved in the access types and algorithms configurations.

FDP_ACC.1 Subset access control

In order to use a container managed by the TOE, the user must present a valid access key, associated with the container. This security requirement is implemented in the TOE by the security functions:

- F.ACCESS_KEY_MANAGEMENT for the configuration of the accesses to the container by the administrator
- F.ACCESS_CONTROL for the access control to the containers.

FDP_ACF.1 Security attribute based access control

In order to use a container managed by the TOE, the user must present a valid access key, associated with the container. In order to implement this operation:

- Rights and roles are associated with users (F.ACCESS_KEY_MANAGEMENT),
- and access to container is therefore controlled (F.ACCESS_CONTROL).

FDP_ITC.1 Import of user data without security attributes

Some data required for the functioning of the TOE are imported from outside of the TSF as access keys or passwords entered by the user. These are data only, no security attribute is imported.

The security function F.SECURE_INPUT implements the communication of data provided as TOE input, and thus covers this requirement.

FDP_RIP.1 Subset residual information protection

This functional requirement is implemented by the security function F.ACCESS_KEY_MANAGEMENT that performs the secure access key erasing in RAM.

FDP_SDI.2 Stored data integrity monitoring and action

This functional requirement is implemented by F.CRYPTO_OPERATIONS that enforces all the operations required for checking the integrity of the files.

FIA_AFL.1 Authentication failure handling

When opening a container, the maximum number of attempts to enter the passwords or PIN is fixed at five. When the number of unsuccessful attempts is reached, the opening request is rejected and the user must repeat the whole authentication process (which causes a slowdown between the different authentication sequences).

The security function F.ACCESS_CONTROL implements this functional requirement.

FIA_UAU.2 User authentication before any action

No container opening is possible on the TOE without a prior phase of user authentication and identification. For each user authentication, users must enter a valid access key.

This functional requirement is implemented by:

- F.TOE_CONFIGURATION to configure the authorized access (access type, password strength, certificate type …),

- F.ACCESS_CONTROLE to enforce access control,
- F.SECURE_INPUT to secure the communication of input data entering the TOE.

FIA_UID.2 User identification before any action

No container opening is possible on the TOE without a prior phase of user authentication and identification. For each user authentication, users must enter a valid access key.

This functional requirement is implemented by:

- F.TOE_CONFIGURATION to configure the authorized access (access type, password strength, certificate type …),
- F.ACCESS_CONTROLE to enforce access control,
- F.SECURE_INPUT to secure the communication of input data entering the TOE.

FMT_MOF.1 Management of security functions behaviour

Only the security administrator can disable or enable the recovery and SOS functions.

The security functions F.TOE_CONFIGURATION associated with F.ACCESS_CONTROL (authentication, recovery and user SOS entry) implement this requirement. Two dedicated security policies (signed by the security administrator) enable or disable the use of the recovery and SOS accesses.

FMT_MSA.1 Management of security attributes

Only security and access administrators are allowed to change the default value, modify, or remove the security attributes "access keys and role".

This security attribute is stored in the control file hidden by Zed!.

The security function F.ACCESS_KEY_MANAGEMENT implements this requirement.

FMT_MSA.2 Secure security attributes

The security function F.TOE_CONFIGURATION (password strength, certificate control for example) ensures that the security attributes “access keys” are secure (these

keys are managed by the standard edition only).

FMT_MSA.3 Static attribute initialisation

The TSF allows the security and access administrators to specify initial alternative values to replace default values when an object or information is created (choosing the role for example).

The security function F.ACCESS_KEY_MANAGEMENT (changing the role for example) implements this requirement.

FMT_MTD.1 Management of TSF data

Only the security administrator is allowed to manage security policies.

This requirement is implemented by the security function F.TOE_CONFIGURATION in charge of verifying the policy signature.

FMT_SMF.1 Specification of Management Functions

The TOE supports the following management functions:

- Access management,
- Recovery (and SOS) operations.

This requirement is implemented by the security functions F.ACCESS_KEY_MANAGEMENT (access and recovery management) and F.TOE_CONFIGURATION (policies).

FMT_SMR.1 Security roles

The TOE supports user, access administrator and security administrator roles.

This requirement is implemented by F.ACCESS_KEY_MANAGEMENT that identifies administrator and user rights depending on their access keys.

8.4. Rationale for Protection Profile conformance claim

This security target does not claim any compliance with a protection profile. No rationale is required.