

E M S C B – Milestone No. I
Secure Linux Hard-Disk Encryption

ANALYSIS SPECIFICATION

based on
European Multilaterally Secure Computing Base (EMSCB)



Abstract: The aim of this EMSCB-based security service is to provide a strongly isolated device encryption for Linux, where the secret key information and all related security-critical operations are not under the control of Linux and furthermore strongly protected and isolated from Linux.

Version June 19, 2006

Contents

1	Architecture Description	2
2	Analysis Model	4
2.1	Analysis Package Identification and Authentication	4
2.1.1	Use-Case Realization of UC 10 User Authentication	4
2.1.2	Use-Case Realization of UC 40 Password Change	4
2.1.3	Use-Case Realization of UC 60 Smartcard Sign-On	5
2.1.4	Use-Case Realization of UC 70 Smartcard Change.	5
2.2	Analysis Package Harddisk Encryption	6
2.2.1	Use-Case Realization of UC 20 Register Hard-Disk	6
2.2.2	Use-Case Realization of UC 30 Deregister Hard-Disk	7
2.3	Analysis Package User and Resource Configuration Manage- ment	7
2.3.1	Use-Case Realization of UC 50 Initialization	7
2.3.2	Use-Case Realization of UC 51 Add User	8
2.3.3	Use-Case Realization of UC 52 Delete User	8
2.3.4	Use-Case Realization of UC 53 User Rights Adminis- tration	9
2.3.5	Use-Case Realization of UC 54 Add Resource	10
2.3.6	Use-Case Realization of UC 55 Delete Resource	10
2.4	Analysis Package Crypto Library	11
2.4.1	Analysis Class Cipher	11
2.4.2	Analysis Class Key	11
2.5	Analysis Package GUI Widget Library	11
2.5.1	Analysis Class GUIServer	12
2.6	Analysis Package Persistent Storage	12
2.6.1	Analysis Class PersistentStorage	12

1 Architecture Description

The Secure Linux Device Encryption (HDD-Encrypter) shall be a task isolated from Linux. Linux on its own cannot provide such capability because the Linux kernel can control the whole hardware and hence all software processes. Once an application process can compromise the Linux kernel it can take over control of the kernel and all other software processes. Since the Linux kernel has a monolithic architecture and very large code size, the probability of software flaws that can be exploited is not negligible.

Furthermore the Linux hard-disk encryption system `dm_crypt` is placed within the kernel. Thus all secret encryption keys would be under the control of the Linux kernel. To separate this relation and to give control over the encryption and secret keys to a separate software process, it is necessary to use a different underlying operating system.

This operating system must provide the ability to isolate processes and should be able to run Linux as one of its processes. Then, the EMSCB HDD-Encrypter can be executed as separated and isolated process. For example, micro-kernel based operating systems provide such options. Another approach would be to use hypervisors, i.e. virtual machine monitors. Figure 1 shows an overview of the architecture for the multi-user scenario. In single-user mode the persistent storage and the GUI server will not be needed, since no configuration data will have to be stored and no configuration menu dialogs will be needed.

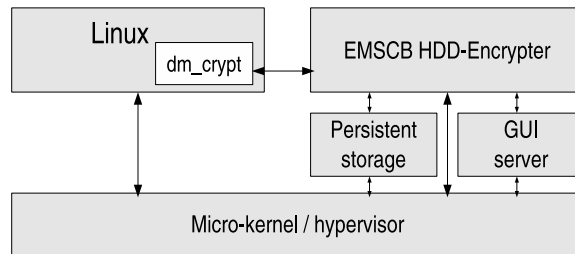


Figure 1: Generic architecture of secure Linux hard-disk encryption.

The HDD-Encrypter itself has the responsibility to encrypt and decrypt the data. In single-user mode the encryption key is always the same. In multi-user mode there will be resources that have to be encrypted with different keys for the purpose of confidentiality. However, there might be resources that are shared by several users but encrypted at the same time. Hence the users have to share the encryption key as well.

Therefore, in multi-user mode we will need classes that deal with user and resource management besides the encryption facility. To be able to distinct between different users and to choose the correct encryption keys we need

classes that are responsible for user identification and authentication. The decomposition of analysis packages of the system can be seen in Figure 2.

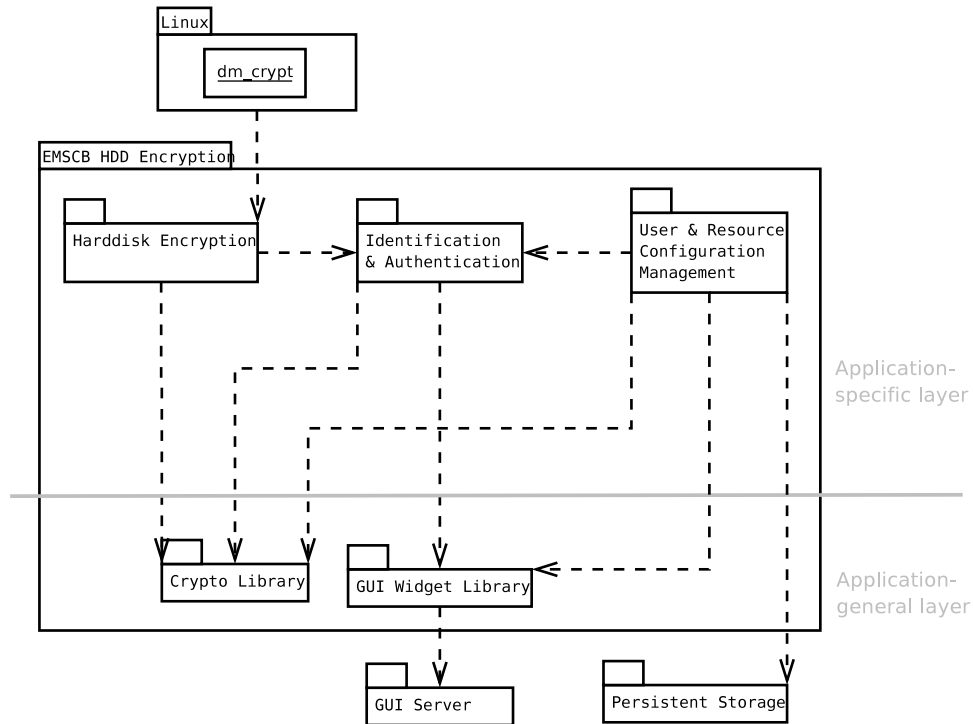


Figure 2: Analysis architecture of the HDD encryption system.

Besides the application specific tasks we can identify responsibilities that also apply to different applications, namely cryptographic routines and graphical user interface widgets. We subsume these tasks within the Crypto Library package and the GUI Widget Library package, respectively.

In multi-user mode the user needs a graphical user interface to enter his identification and authentication data, and the administrator needs an interface to configure users, resources and their relationships, i.e. access rights.

Besides the hard-disk encryption the configuration management will need cryptographic functions to protect the encryption keys, i.e. by encrypting them. Authentication schemes usually need cryptographic functions as well, e.g. hash functions. Thus we have dependencies from the major analysis packages of the HDD-Encrypter to the Crypto Library.

The Configuration Management package depends on the Persistent Storage package, since it needs to persistently store the user and resource configuration data. Moreover, the GUI Widget Library depends on some kind of GUI server, which has to be provided by the operating system in order

to display the widgets and provide an event mechanism.

2 Analysis Model

2.1 Analysis Package Identification and Authentication

2.1.1 Use-Case Realization of UC 10 User Authentication

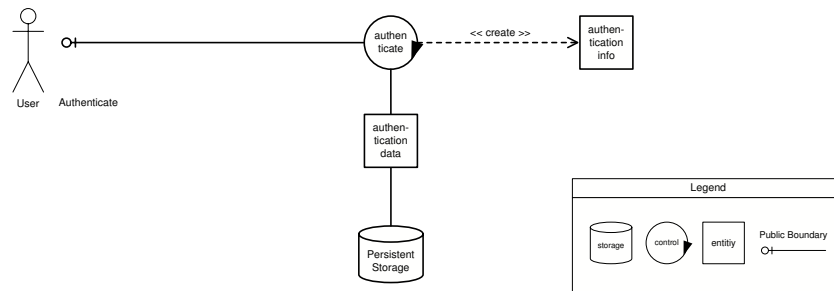


Figure 3: Class diagram for the *User Authentication* use-case realization

A **user** chooses to authenticate himself to prove its identity. The **authenticate** control object asks the **user** for username (within a multiuser environment) and its corresponding authentication data (password). The **authenticate** control object compares the entered authentication data with the **authentication data** entity object from the persistent storage. If the **authenticate** control object could verify user's authentication data, it creates a **authentication info** entity object and passes it over to the **registration** control object.

2.1.2 Use-Case Realization of UC 40 Password Change

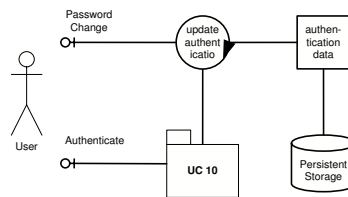


Figure 4: Class diagram for the *Password Change* use-case realization

A **user** requests to change his authentication data using the **Password Change** interface. Therefore the **update authentication** control object uses **package UC 10** to verify the authorization of his intention using the

Authentication interface. After successful authentication the **user** can renew his **authentication data** object. If the **user** fails the authentication or cancels his intention, **update authentication** prevents changing the corresponding **authentication data object**.

2.1.3 Use-Case Realization of UC 60 Smartcard Sign-On

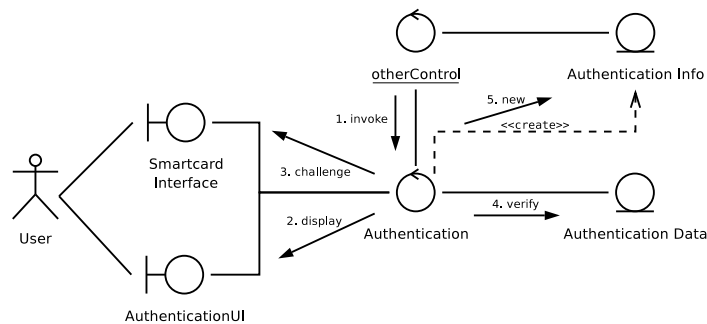


Figure 5: Collaboration diagram for the *Smartcard Sign-On* use-case realization

If an user authentication must be performed according to another use case, the respective use case control object invokes the **Authentication** (1).

The **Authentication** control object displays the **Authentication UI** so that the user can identify himself (2).

After identification the **Authentication** starts a challenge-response protocol with the smartcard using the **Smartcard Interface**. The **User** has to insert his smartcard in the smartcard reader so that the response can be computed (3).

The **Authentication** verifies the response received from the **Smartcard Interface** object by using a previously stored **Authentication Data** object (4).

If the authentication can be verified, the **Authentication** will create a new **Authentication Info** object, which the other use case control object use to authorize the user (5).

2.1.4 Use-Case Realization of UC 70 Smartcard Change.

Some open questions:

- Who will change the smartcard, i.e. user or administrator?
- When and under what circumstances will this happen?
- Are updates of the authorization data necessary after a change?

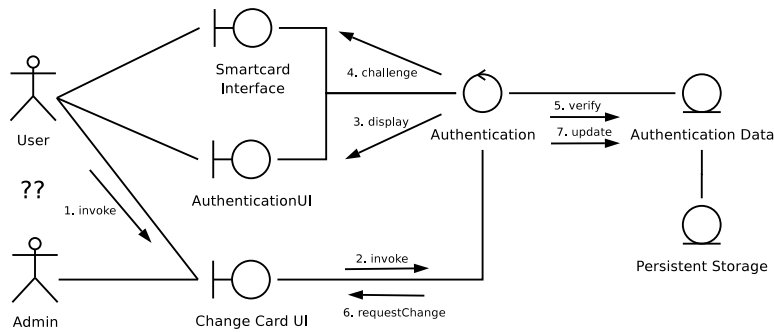


Figure 6: Collaboration diagram for the *Smartcard Change* use-case realization

- Will the ID of the user remain the same or are the public key on the smartcard and the ID the same?

Use case needs further requirements capture and analysis!!

2.2 Analysis Package Harddisk Encryption

2.2.1 Use-Case Realization of UC 20 Register Hard-Disk

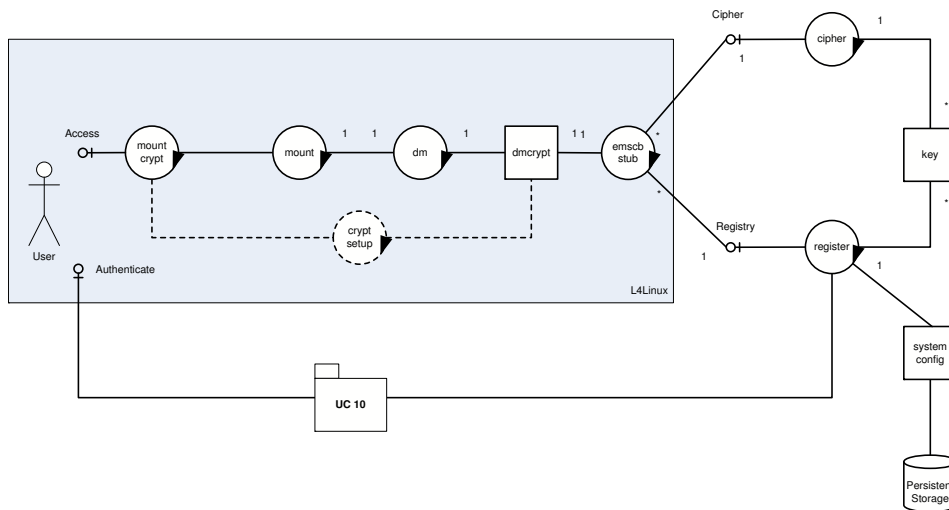


Figure 7: Class diagram for the *Register Hard-Disk* use-case realization

A **user** requests to register his encrypted hard-disk partitions (specified by the **resource id**) via **mount crypt**. The settings for **mount** and **dm** are

provided using **crypt setup** control object once-only before. The **dmccrypt** control object in turn uses the **emscb stub** to communicate and transfer all corresponding data between L4Linux and the L4 hard disk encryption task. After an initially request on the encrypted block device using the **register interface** together with the provided **resource id**, the **UC 10 package** will be used to authenticate the **user** and ask for the necessary authentication info. If the **user** authentication fails the system refuses access to requested encrypted partition. After a successful user authentication, **register** checks the necessary access rights written in **authorization data** entity object and provides the corresponding **key object** if user has the appropriate authorizations. Only then the **cipher** control object encrypts and decrypts all incoming respectively outgoing block data over the **cipher interface**.

2.2.2 Use-Case Realization of UC 30 Deregister Hard-Disk

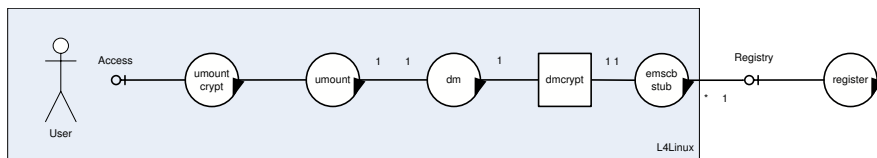


Figure 8: Class diagram for the *Deregister Hard-Disk* use-case realization

A **user** requests to deregister his encrypted hard-disk partitions. The **umount crypt** control object ask for acknowledgment of his intention. After the positive acknowledgment the **umount crypt** process will then use the following control objects (umount, dm, emscb-stub, register) to deregister the user from all encrypted partitions that belong to the user, while updating the **dmccrypt** entity object.

2.3 Analysis Package User and Resource Configuration Management

2.3.1 Use-Case Realization of UC 50 Initialization

For system initialization, the **Init** control displays the **Init UI** which informs the **Administrator** about the first-time initialization. The **Init UI** uses the **Create User UI** which request the input of username and password. The **Add User** control creates a **User Data** object which is passed to the **System Configuration** to be added to the configuration in the **Persistent Storage**. Now the **Init** control adds **Access Rights** containing the administration flag to the **System configuration**. Finally a confirmation message is displayed to the **Administrator**.

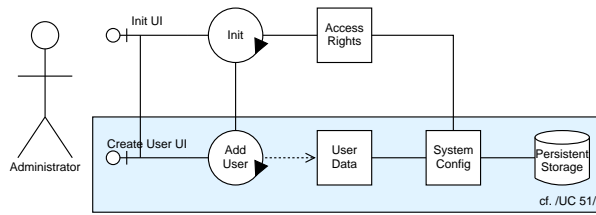


Figure 9: Class diagram for the *Initialization* use-case realization

2.3.2 Use-Case Realization of UC 51 Add User

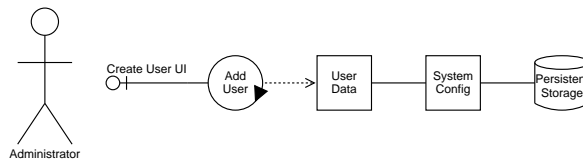


Figure 10: Class diagram for the *Add User* use-case realization

An **Administrator** requests to add a new user. The **Add User** control uses the **Create user UI** to request the input of username and password and creates a **User Data** object containing the input information. The **User Data** is forwarded to the **System Configuration** which checks it for uniqueness in the configuration data provided by **Persistent Storage**. If the **User Data** is not unique, an error message is displayed. Otherwise the **User Data** is written to the **Persistent Storage** and a confirmation message is displayed to the **Administrator**.

2.3.3 Use-Case Realization of UC 52 Delete User

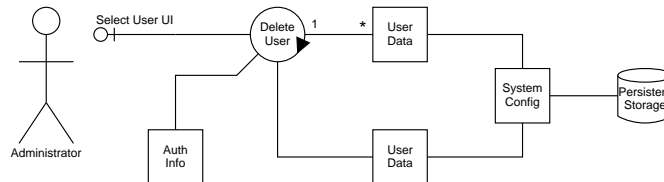


Figure 11: Class diagram for the *Delete User* use-case realization

An **Administrator** requests to delete an existent user. The **Select User** control request all **User Data** elements from **System Configura-**

tion and uses the **Select User UI** to display a list of all known users. The **Administrator** selects a user. The **Select User** control tests if the selected user is not equal to the currently authenticated **Administrator** in **Auth Info** coming from /UC 10/. If the selected user is equal, an error message is displayed. Otherwise, a warning message is displayed which must be acknowledged by the **Administrator**. If the warning message is accepted, the selected **User Data** object is passed to **System Configuration** which removes the requested user from the configuration database stored in **Persistent Storage**.

2.3.4 Use-Case Realization of UC 53 User Rights Administration

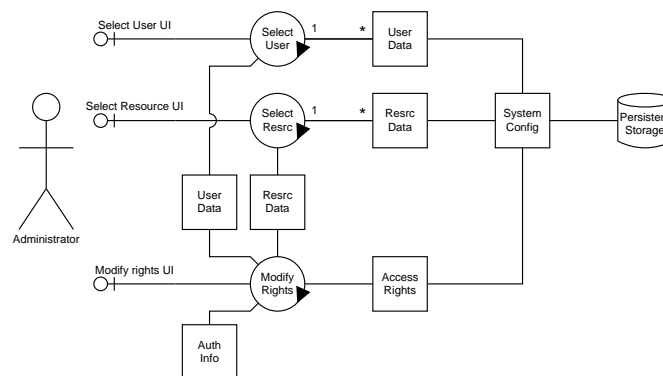


Figure 12: Class diagram for the *User Rights Administration* use-case realization

An **administrator** request to change the access rights for a user. The **Select User** control requests all **User Data** elements from the **System Configuration** and displays it using the **Select User UI**. The **Administrator** now selects a user identified by **User Data** which is forwarded to the **Modify Rights** control. The **Select Resource** control requests all **Resource Data** elements from the **System Configuration** and displays it using the **Select Resource UI**. The **Administrator** now selects a resource identified by **Resource Data** which is forwarded to the **Modify Rights** control. Now the **Modify Rights** control request the current **Access Rights** for **User Data** and **Resource Data** from the **System Configuration**. The **Modify User UI** then displays the **Access Rights** and allows the **Administrator** to change it. When the **Administrator** has confirmed the requested modifications, **Modify Rights** uses the **Auth Info** from /UC 10/ to check if the administration flag shall be removed from the currently logged in **Administrator**. If the check is true, an error message is displayed. Otherwise, the updated **Access Rights** are passed

to **System configuration** which stores the new information in the **Persistent Storage**. Finally the **Modify Rights** control shows the new **Access Rights** in a confirmation message.

2.3.5 Use-Case Realization of UC 54 Add Resource

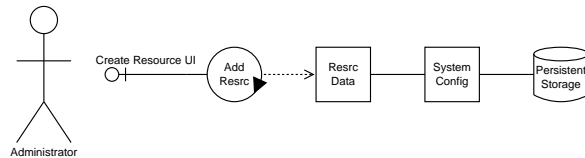


Figure 13: Class diagram for the *Add Resource* use-case realization

An **Administrator** requests to add a new resource. The **Add Resource** control uses the **Create Resource UI** to request the input of the resource name and creates a **Resource Data** object containing the input information. The **Resource Data** is forwarded to the **System Configuration** which checks it for uniqueness in the configuration data provided by **Persistent Storage**. If the **Resource Data** is not unique, an error message is displayed. Otherwise the **Resource Data** is written to the **Persistent Storage** which requests the desired binding (bind to hardware and/or software components) and displays a confirmation message upon success.

2.3.6 Use-Case Realization of UC 55 Delete Resource

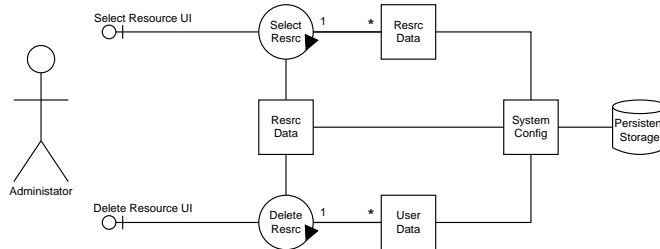


Figure 14: Class diagram for the *Delete Resource* use-case realization

An **administrator** request to delete an existent resource. The **Select Resource** control requests all **Resource Data** elements from the **System Configuration** and shows the list using the **Select Resource UI**. The **Administrator** then selects a resource identified by **Resource Data**. The **Resource Data** is forwarded to the **Delete Resource** control which requests all **User Data** elements from **System configuration** that currently

have access rights on on the selected resource. The user list is displayed using the **Delete Resource UI** and must be acknowledged by the **Administrator**. If the **Administrator** acknowledges the **Delete Resource UI**, the **Delete Resource** control passes the **Resource Data** to **System Configuration** which removes it from all user accounts and stores the updates in **Persistent Storage**. Finally a confirmation message is displayed.

2.4 Analysis Package Crypto Library

The Crypto Library package mainly contains the **Cipher** and **Key** classes which are responsible for encryption and decryption of the data stored on the hard-disk.

2.4.1 Analysis Class Cipher

Responsibilities This class is responsible for encryption and decryption of data stored on partitions of the hard-disk.

Attributes

- algorithmType

Associations This class is associated with a **Key** (2.4.2) of certain length.

Special Requirements An implementation of the AES algorithm must be provided. But this class shall probably provide an interface for using different algorithms.

2.4.2 Analysis Class Key

Responsibilities The Key is used for encryption and decryption of data stored on a certain partition of the hard-disk.

Attributes

- keylength

Associations This class is associated with the **Cipher** class (2.4.1) and the **Register** class, whereas a **Register** object can be associated with several **Key** objects. But a **Key** object is related to only one **Register** object and one **Cipher** object.

2.5 Analysis Package GUI Widget Library

The GUI Widget Library package contains abstractions for widgets and event handling of the graphical user interface mechanism.

2.5.1 Analysis Class GUIServer

Responsibilities This class is responsible for communication with the underlying GUI mechanism, which is illustrated by the GUI Server package. It must provide a way to send commands compatible to the underlying server mechanism to display GUI widgets.

Associations This class is associated with all boundary classes that represent a graphical user interface, i.e. from the User and Resource Configuration Management package (cf. 2.3) and the Identification and Authentication package (cf. 2.1).

Special Requirements This class must provide an abstraction for the GUI Server package. There shall be abstractions for the different GUI widgets and this class must be able to work with these abstractions as well.

2.6 Analysis Package Persistent Storage

2.6.1 Analysis Class PersistentStorage

Responsibilities This class is responsible for storing data persistently and additionally binding it to

- a user authorization secret,
- and/or software components (e.g. an application),
- and/or hardware components (e.g. a TPM).

User interaction is required to determine the desired binding (if any).

Attributes

- object to store
- bit mask to define binding

Associations The PersistentStorage class uses the TPM Server class to invoke TPM commands.

Special Requirements The PersistentStorage class must be able to access the hard-disk by means of the underlying operating system.